Table of Contents

Video Display Connection

There are three different methods of attaching a video display to the C4P computers. These are outlined as follows:

1. Preferred method - connect the supplied computer video cable to the high impedance (Hi-Z) input of a closed-circuit TV video monitor. Ohio Scientific offers a color television set, modified for video monitoring. Ohio Scientific also offers the Model AC-3 12" black and white monitor. Both are ideal for this application. The units double as a television when the video cable is disconnected.

2. Connect the supplied computer video cable to an "RF modulator" which is, in turn, connected to a standard television's antenna terminals. RF Modulators are inexpensive and allow you to use almost any television with your computer. They are sold in kit form.

3. Have a standard AC transformer-operated television modified to accept direct video entry. This requires special safety precautions which will be explained later.

...osed-Circuit Video Monitor Connection

1. Refer to Figure 1. Attach the supplied video cable to the computer as shown.

2. Connect the other end of the cable to the high impedance input of the video monitor. The AC-3 monitor has a Hi-Z RCA-type phono jack input. On other monitors, a high impedance - low impedance selector switch is sometimes present, or there may be two or more inputs. Consult the manufacturer's instructions.

3. Observe the manufacturer's power recommendations. If the monitor has a 3-wire grounded plug, connect it to a properly grounded 3-wire AC outlet.

4. Turn on the computer and monitor.

5. Allow the monitor to warm-up. You should see the screen filled with random graphics characters, alphabet, etc.

6. If necessary, adjust the VERTICAL and HORIZONTAL controls to obtain a stable picture.

RF Modulator/Standard TV Connection

1. Refer to Figure 1. Review the manufacturer's instruction included with the RF modulator.

2. Connect the computer video cable to the computer as shown.

3. Connect the video cable to the RF Modulator.

4. Connect the modulator to the television's antenna terminals (consult modulator instructions).

5. Plug in the television and computer.

6. Turn on the computer, television, and modulator (consult modulator instructions).

7. At this point you will have to select the proper TV channel and possibly adjust the television's fine tuning slightly (consult modulator instructions).

8. When the television warms up you should observe a screen filled with random graphics characters. If the picture is not stable, adjust the television's VERTICAL or HORIZONTAL controls as needed.

Modification of a Television For Direct Video Entry

1. A standard television may be modified to act as a video monitor. However, this conversion requires detailed knowledge of television circuitry, and will likely require a schematic of the television to be converted. Consult a qualified service person.

WARNING

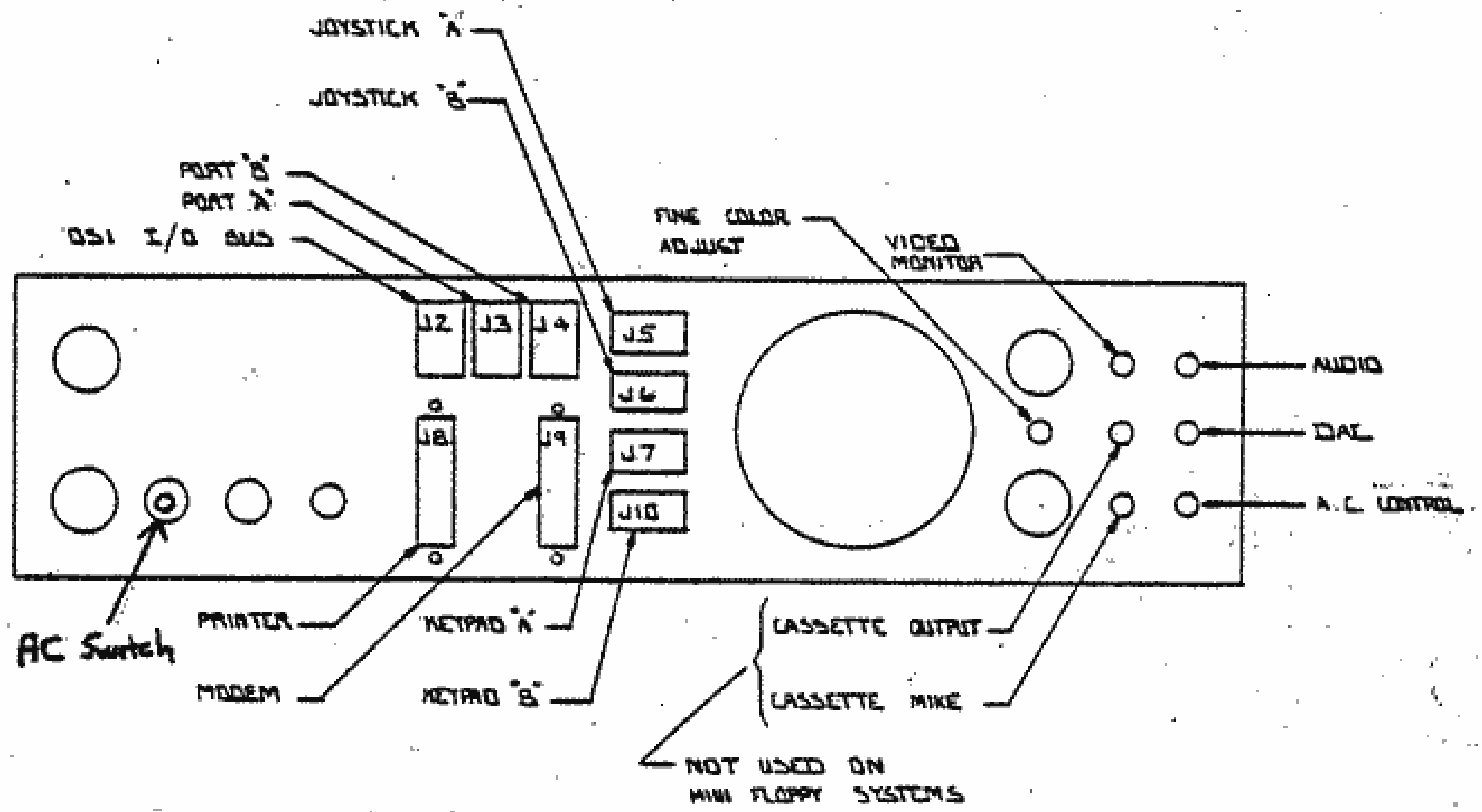
ANY TELEVISION CONVERSIONS MUST BE PERFORMED ONLY BY A QUALIFIED PERSON, SUCH AS A TV SERVICEMAN. LETHAL VOLTAGES ARE PRESENT WITHIN THE TELEVISION. INCORRECT CONNECTIONS MAY PRESENT SHOCK HAZARDS OR DAMAGE THE COMPUTER. SUCH DAMAGE IS NOT COVERED BY THE WARRANTY.

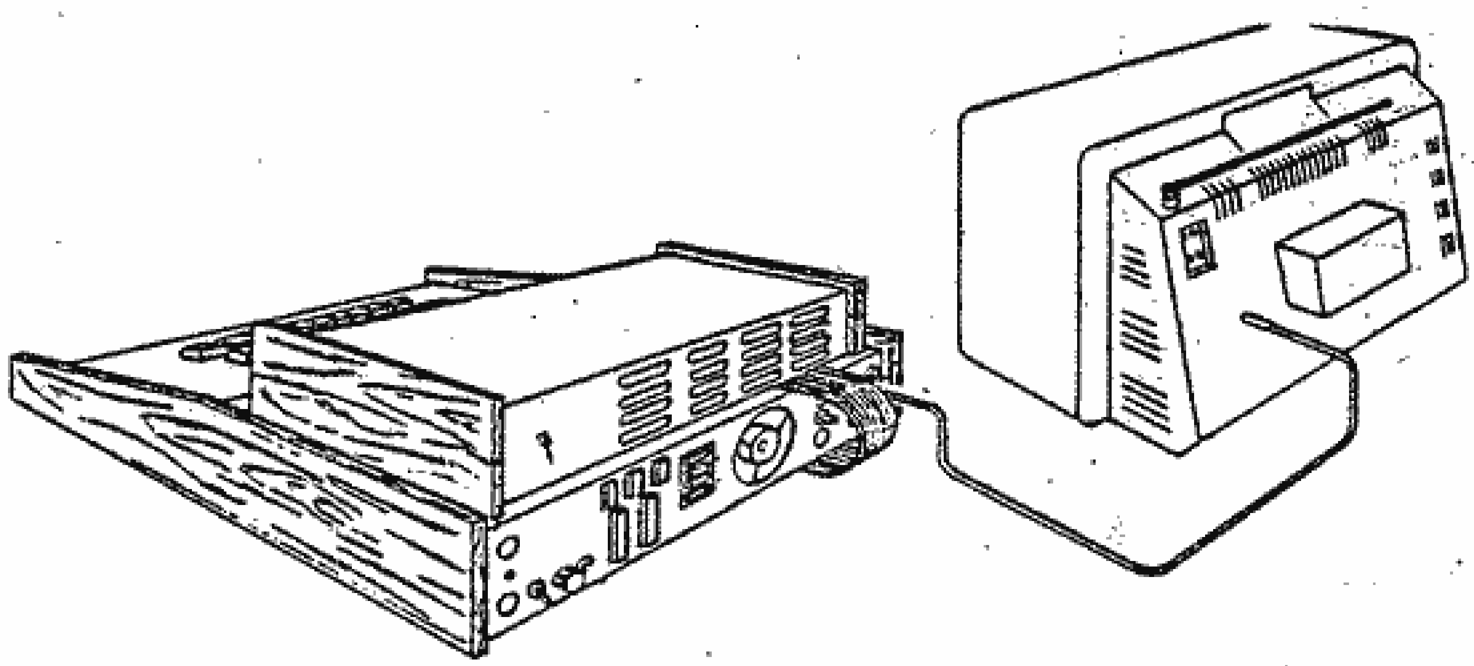
2. The television to be modified must be an AC-transformer operated television. Several solid-state TV sets are now available which can be operated from 110V AC, or from a 12 volt source such as a car cigarette lighter. These televisions can usually be converted easily. Some older "AC-DC" tube-type televisions are "hot cha types; that is, one side of the power line is conne to the chassis. These televisions do not have tra and MUST NOT be used for conversions.

3. More characters per line can be displayed on the screen if the picture is "shrunken" slightly. On most 11ØV AC/ 12V DC televisions, this can be accomplished by adjusting the television's power supply regulator to give a lower voltage. Brightness will also be diminished, but this can be restored via the TV BRIGHTNESS CONTROL. Refer this adjustment to the service person at the time of conversions.

4. When the power supply voltage is adjusted, the picture may require re-centering. Equal borders can be restored to the screen by adjusting the picture tube centering coils. Refer this adjustment to the service person at the time of conversion.

5. When the television has been modified, it may then be treated as a video monitor and connected to the computer. Refer to that section of this manual.





4. Starting Up Your System - Demo Disks

Now you are ready to power up your C4 System.

Power Up

1) Check that your system is connected according to Figure 1 and the related instructions. Make sure that there is clearance for ventilating air in the back of the C4P system.

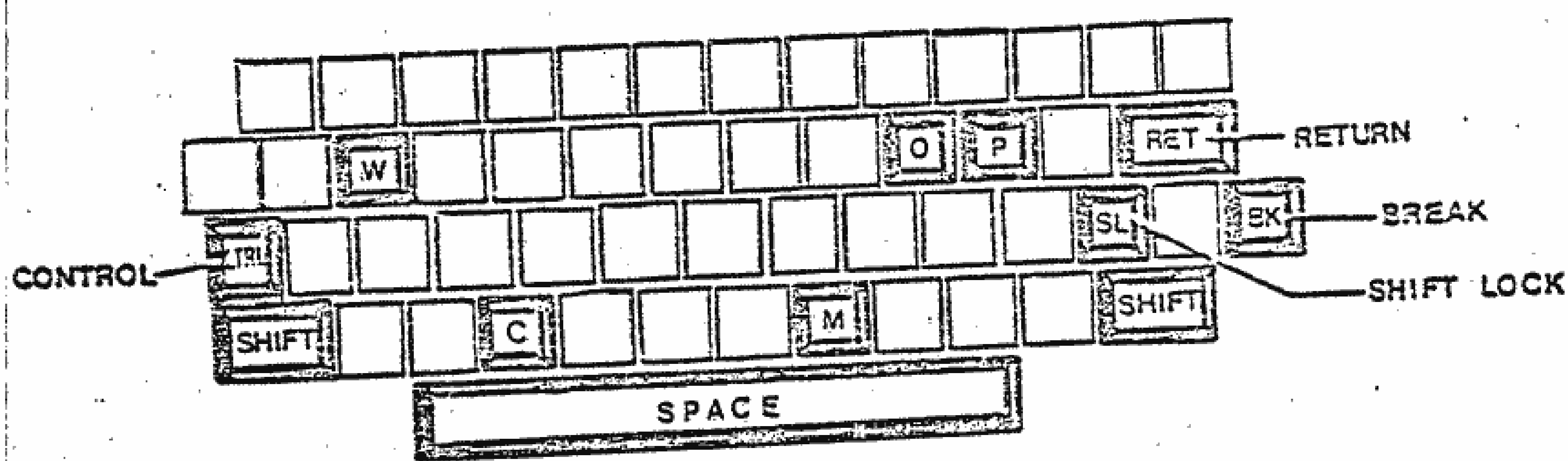
2) Plug in power cords.

3) Turn on power on the back of your keyboard console.

4) Turn on floppy disk power (switch is on rear of disk drive).

5) Turn on CRT and any other accessories.

6) Depress the SHIFT LOCK key. Now press the "BREAK key on the keyboard.



7) Remove the disk labeled "Customer Demo Disk" from its covering sleeve. Carefully insert the disk with your right thumb on the label. Keep the disk label on the top side.

The disk should be inserted firmly until a click is heard or slight resistance is encountered. Close the door on the disk drive. The light will not normally come on, as the disk has not yet been selected by the computer.

8) MAKE SURE THE "SHIFT LOCK" KEY IS DEPRESSED. When the computer responds "H/D/M?" on the CRT (television screen), type

D

The program will automatically be loaded into the computer from the disk.

This disk will repeat its program endlessly.

Disk Programs

The Customer Demo Disk contains a continuously sequenced animation, showing the power of the OSI C4P computer and its software. In this manual, we shall show you how to adapt some of these programs to your special purposes. Similar programs are available from your OSI dealer. When you are finished, remove the disk from the drive and store the disk in its protective sleeve. If you wish to use another disk, press

<BREAK>

and then insert the new disk in the disk drive, then repeat Step 7 of the previous section.

The "Dealer Demo Disk" contains the programs

1) Graphics Demo, an image generator which shows the tools of animation and graphing.

2) Plane Banner, a simulated airplane made from the C4P's Character set. A wide variety of shapes is possible.

3) Random Square, an animated pattern generator to show the color range available.

4) Kaleidoscope, a continuously changing pattern to illustrate the variety of symbols available.

5) Space Wars, a game to pit your starship against the enemy empire.

6) Hectic, a ricochet simulation game. Both scientific problem simulation and games can use these techniques.

7) Tiger Tank, a combat game to show real-time player interaction.

8) Set Time, a clock function which does more than keep time. This program can be used to control other programs.

9) AC Demo, a home light and appliance control program. With the external lamp modules attached, the pictures on the CRT screen will be echoed by the device behavior. (Note that remote module switches must be properly set to use this program.)

These programs can be readily adapted to your use. When you are familiar with your C4P system, you will be able to list these programs and extract the examples for your special purposes. These well written examples provide programming lessons and power for sophisticated programs.

Power Down

When you are ready to turn the system off:

1) Remove the disk from the disk drive by pushing the rectangular button below the disk door. Then remove the disk, placing it back in its sleeve.

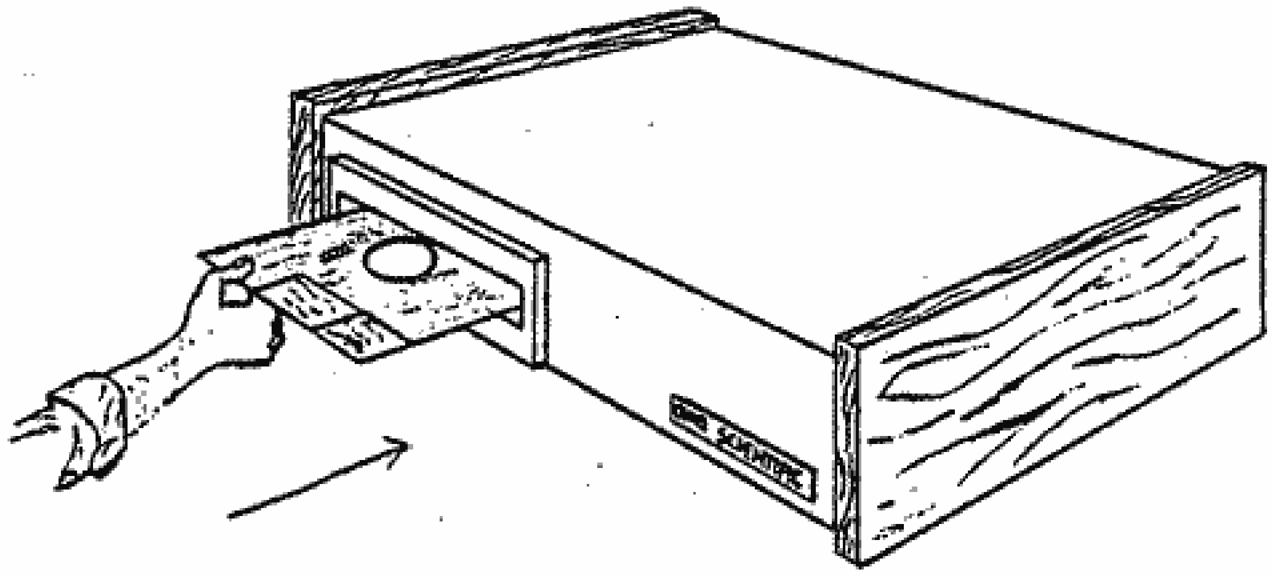
2) Turn off peripheral devices, if any.

3) Turn off CRT.

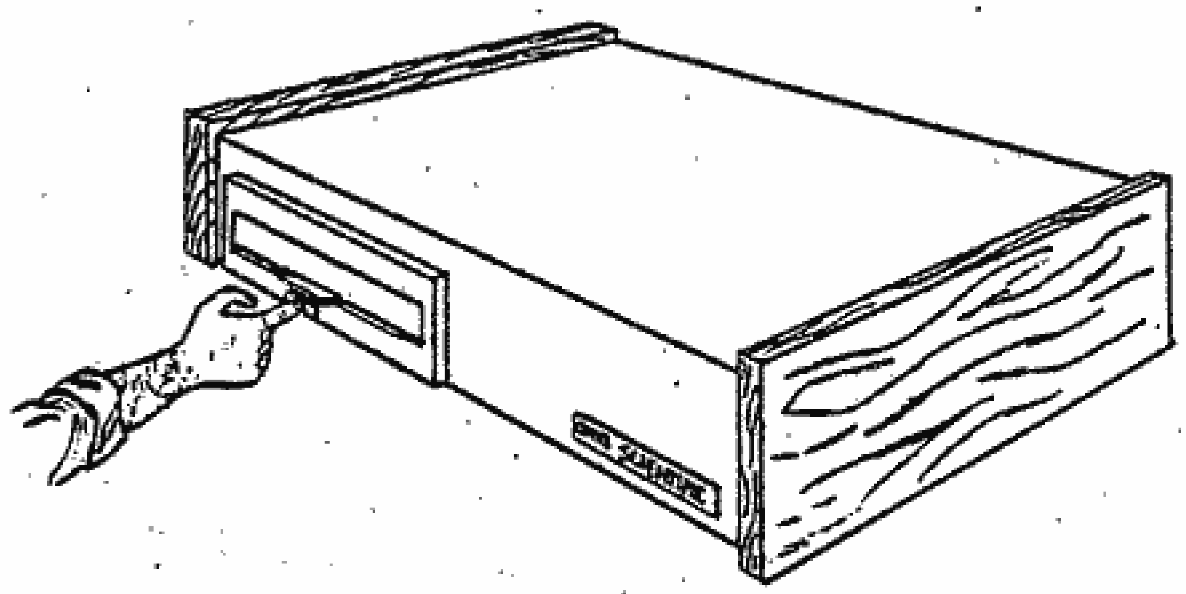
4) Turn off disk drive computer.

5) Turn off computer power (back of keyboard console) last.

You have just completed the hardest part of using your computer! From here, your care of the computer and orderly handling of materials will pay itself back in reliability and enjoyment of the C4P system. Let's go on to using your system in some applications!



Inserting a Disk.



To remove a Disk.

5. Notation

In order to discuss programs with you, we need a common notation.

We shall use the shorthand notion

<RETURN>

instead of saying "Press the "RETURN" key". Do not type the brackets or the word RETURN letter-by-letter.

Blank spaces will be indicated by a blank in our typing, such as

1Ø GOTO 5 <RETURN>

rather than write

1Ø <SPACE> GOTO < SPACE > 5 <RETURN>

When we want you to enter something from the keyboard, your responses will be underlined or in brackets (the messages produced by the C4P will not be underlined). In the following example,

FUNCTION?

UNLOCK <RETURN>

The C4P asked the question "FUNCTION?" and your response would be to type out "UNLOCK" (note that all of the letters are capitalized) and then a carriage return.

Chapter II

BASIC Programming

You have used the applications programs provided on the customer demo disk to demonstrate the power of your OSI C4P system. You will often want to to write your own programs in a powerful but simple language. BASIC is such a language.

An excellent book by Dwyer and Critchfield, BASIC and the Personal Computer, is available from your OSI dealer. However, we should not hesitate to try out some simple programs. This section is not intended to cover all of BASIC. Instead, it is to show extensions and differences of OSI's BASIC that the user should know. A few simple examples are included to familiarize the new users with applications.

First, please turn on your OSI C4P computer. Remember

1. Turn on the computer power first and second the floppy disk's power (power switches are located on the rear panel see Figure 1).
2. Turn on the video display console.
3. Press <BREAK>.
4. Insert your minifloppy disk marked simply "OS-65D V31.".
5. Verify that the shift lock key is down. Press D on the keyboard.
6. Respond to the question

    FUNCTION?

    by typing

    UNLOCK <RETURN> .

    (We'll underline your entries for emphasis.)

Now clean out the work space (memory where your program is running) by responding to the BASIC prompter

OK

NEW <RETURN>

This will erase the old programs which occupied the available memory. Next type

LIST <RETURN>

to verify that no programs are present.

1. Calculator Mode (Immediate Mode)

Let's try our BASIC in the easiest form, first. Type the line below

PRINT 5+3 <RETURN>

(Remember underlined quantities are entered by you.) The computer will return the answer

8

For brevity, we can also use the question mark, "?", in place of PRINT as

? 5+3 <RETURN>

The result is the same. This calculator-like function is called the immediate mode of operation. You can use it like a scientific calculator.

Program Mode

Let's repeat this program with the input and the output controlled by the computer (program mode). Type

10 ? 5+3 <RETURN>

or

10 PRINT 5+3 <RETURN>

Since we have started the line with a number, the computer will await any further numbered lines before performing the required calculations. This is your first program or set of instructions (in BASIC)! When we are ready to have the calculations run, we then type

RUN <RETURN>

The C4P will now execute the one line program that you just entered. The answer is, as before,

8

We can use the numbering of lines (also called "labeling" for "statements") to perform many instructions consecutively. It is a good practice to number statements as 1Ø, 2Ø, 3Ø, ..., leaving room for easy future addition of lines. You should be careful to arrange the lines in the order we wish them performed. We shall improve the clarity and the usefulness of the previous program by allowing input to the computer when the program is run.

To prompt the program user, we place quotation marks around words which we wish printed on the CRT when the statement is performed. The name of the variable which is to be entered follows the prompting quote, separated by a semi-colon.

Intermediate variables, with convenient names (which do not include words reserved for use by BASIC, such as FOR and WAIT - see the appendix) should be chosen to keep the program statements simple. The final statement, 5Ø, which ends our program indicates to the computer that this is the end of our program.

Let's write out these changes in a program. Type

```
1Ø  INPUT "ENTER THE FIRST NUMBER";A <RETURN>
2Ø  INPUT "ENTER THE SECOND NUMBER";B <RETURN>
3Ø  SUM=A+B <RETURN>
4Ø  PRINT "THE SUM IS";SUM <RETURN>
5Ø  END <RETURN>
```

When you type

RUN <RETURN>

the message in between quotes in line 1Ø will appear as

ENTER THE FIRST NUMBER?

The BASIC program follows the message by a ? to indicate an operator entry is expected. You should respond by typing a number, then a <RETURN>, such as

5 <RETURN>

The computer will inquire again

ENTER THE SECOND NUMBER?

Type your second number in the same manner, such as

3 <RETURN>

The computer will respond by printing

THE SUM IS 8

You should now type

RUN <RETURN>

The computer will again RUN your program and ask you for numbers.

We have seen from the above examples that the BASIC language is algebraic in form, with simple input and output form. By numbering the statements, we have arranged the order of execution of program statements. Upon typing

RUN <RETURN>

the ordered sequence of statements is executed. We note that the words appearing between the quotation marks will be printed on the CRT screen as prompting statements.

Multiple calculations can be performed by using loop statements. For example, computation of the squares of the numbers from 1 to 6 inclusive could be done by the following program (I'll drop writing <RETURN> for simplicity)

```
10  REM SQUARES OF NUMBERS PROGRAM
20  FOR I=1 TO 6      For I=1 to 100
30  SQ=I*I
40  PRINT "THE SQUARE OF";I;"IS=";SQ
50  NEXT I
60  END
RUN
```

I've stopped writing <RETURN> at the end of each line explicitly to make the program look less cluttered. You will still enter <RETURN> when entering the program from the keyboard. If we had used

```
30  SQ=I^2
```

instead, (the up-arrow is entered by <SHIFT N>), we would find slight variations in the answers. This is due to the algorithm (method of calculation) which our BASIC uses. The up-arrow, ^ , means "to the power of." This involves the use of logarithms, instead of merely multiplying. If you type

```
30  SQ=I^2
```

the old statement 30 will be replaced with the new statement 30; then the program will run.

To do a computation until a desired value is found, we can use the less than, greater than, or equal (< , >, =) signs. An example might be to find the smallest integer whose square exceeds 600.

```
10  REM FIND THE INTEGER X SUCH THAT
20  REM (X-1)^2 IS <600 AND
30  REM (X^2) IS >600
40  X=1
50  SQ=X*X
60  IF SQ>600 THEN GOTO 90
70  X=X+1
80  GOTO 50
90  PRINT "THE LOWEST INTEGER X WITH X^2>600 IS";X
100 END
```

Statement 60 is a conditional statement. If it is satisfied, i.e., SQ > 600 is true, then the next statement to be executed is number 90. If SQ > 600 is false, the next statement in order, number 70, is executed. This branching between statements permits a program to be modified, depending on the result of a calculation. This branching technique makes high speed decisions possible, based on the data which is evaluated by the computer. When the conditional branch to statement 90 is made, the answer is then printed.

2. Character Manipulation

In addition to handling numbers, our BASIC (Microsoft 9½ Digit BASIC) can also be used to manipulate characters. For example, to read in a string of characters, we can use

```
10  INPUT "YOUR CHARACTERS ARE";A$
```

The dollar sign after the variable name implies that this is a character string, rather than a number, per se.

Several character string operations are possible. We can print out the characters by

```
20  PRINT A$
```

If we were to run the program at this point (by typing RUN), the reply to

```
YOUR CHARACTERS ARE ?
```

by typing

```
NOW <RETURN>
```

would result in the print out

```
NOW
```

If we had typed

```
NOW IS THE TIME <RETURN>
```

the character string

```
NOW IS THE TIME
```

would have been printed. This last string consists of 12 letters and the three blanks in between words. We can operate on these strings with string operations.

One of the possible string operations is counting the string length

```
3Ø  L=LEN(A$)
```

Therefore, the program

```
1Ø  INPUT "WHAT ARE YOUR CHARACTERS";A$
2Ø  PRINT A$; "WERE READ IN"
3Ø  L=LEN(A$)
4Ø  PRINT "THERE WERE" ;L; "CHARACTERS"
5Ø  END
```

will read in your character string, echo the characters for verification, and print the character count. (BASIC expects 72 or less characters to be input at any time.) Entering "LONG" will echo "LONG" and report four characters.

Other useful string operations are pcking out the left most I characters in a string. For example, the left most character in the string A$ is found via

```
10  L$=LEFT$(A$,1)
```

The two lefthand characters in the string A$ are

```
10  L$=LEFT$*A$,2)
```

Similarly, the right most two characters in the string A$ are

```
10  R$=RIGHT$(A$,2)
```

Likewise, the midrange I characters which start from the Jth one are

```
M$=MID$(A$,I,J)
```

Thus, the second, third and fourth characters of the string A$ are given by

```
M$=MID$(A$,2,3)
```

For example, the program

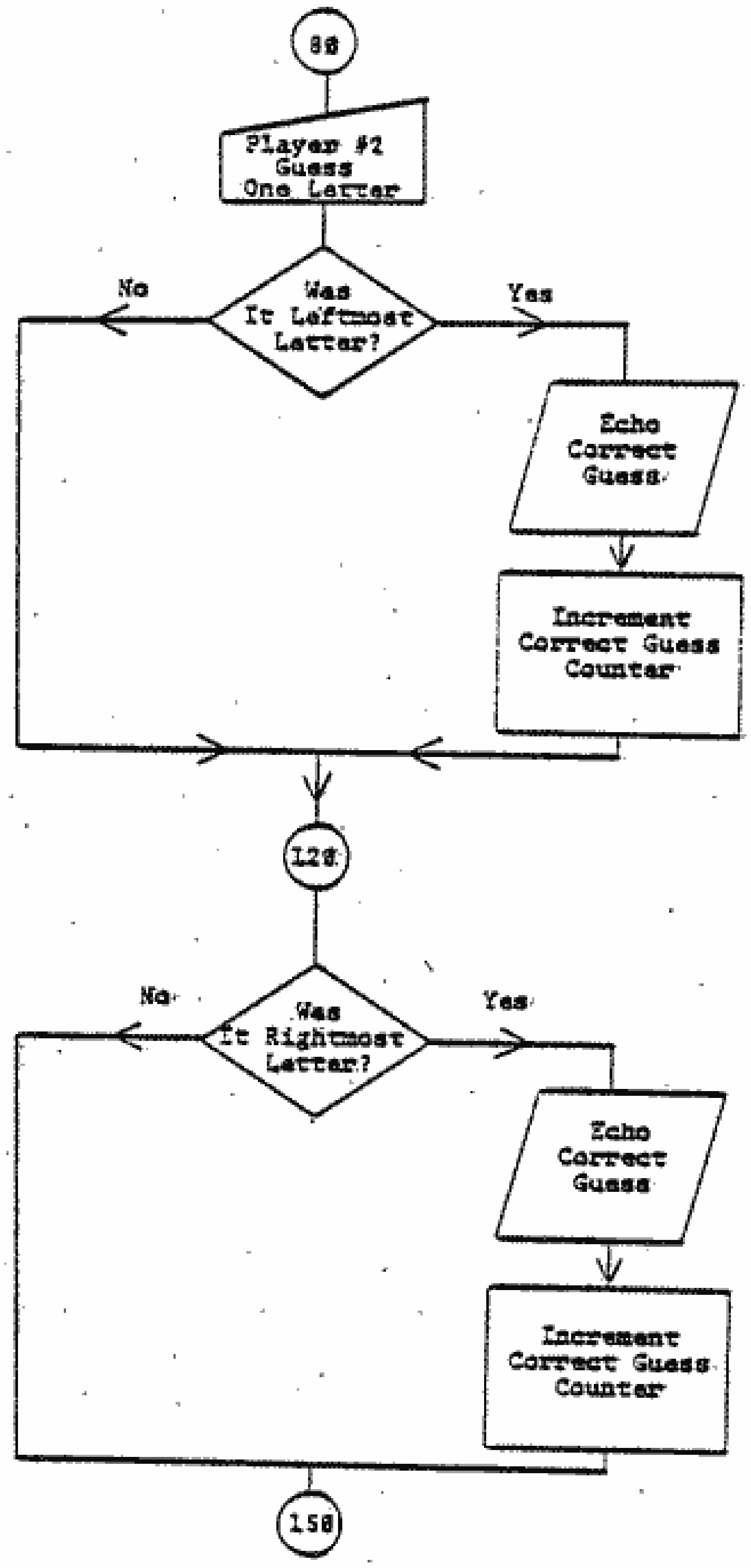
```
10  A$="FRIDAY"
20  PRINT MID$(A$,3,2)
```

will result in the output

```
RID
```

Now we have enough information to write a simple two person hangman type game. Let the first person type a three letter word. The computer will then erase the screen. The second person will try to guess the letters. If the player fails to guess in six tries, the first player wins. OK?

```
10  REM GUESSING GAME
20  INPUT "PLAYER #1 ENTER A 3 LETTER WORD";A$
30  FOR I=1 TO 32 : REM CLEAR
40  PRINT         : REM THE
```



80
Player #2
Guess
One Letter
Was
It Leftmost
Letter?
No
Yes
Echo
Correct
Guess
Increment
Correct Guess
Counter
120
Was
It Rightmost
Letter?
No
Yes
Echo
Correct
Guess
Increment
Correct Guess
Counter
150

```
50  NEXT I            : REM SCREEN
60  COUNT=0           : REM COUNT IS CORRECT GUESS COUNTER
70  TURN=0            : REM TURN COUNTS TOTAL GUESSES
80  INPUT "YOUR ONE LETTER GUESS IS";B$
90  IF LEFT$(A$,1)=B$ THEN PRINT LEFT$(A$,1)
100 IF LEFT$(A$,1)=B$ THEN COUNT=COUNT+1
120 IF RIGHT$(A$,1)=B$ THEN PRINT RIGHT$(A$,1)
130 IF RIGHT$(A$,1)=B$ THEN COUNT=COUNT+1
150 IF MID$(A$,2,1)=B$ THEN PRINT MID$(A$,2,1)
160 IF MID$(A4,2,1)+B$ THEN COUNT=COUNT+1
170 TURN=TURN+1
180 IF COUNT=3 THEN GOTO 300
190 IF TURN=6 THEN GOTO 600
200 GOTO 80
300 PRINT "YOU WIN, THE WORD WAS";A$
310 GOTO 700
600 PRINT "YOU LOST, THE WORD WAS";A$
700 END
```

Of course, if we got one letter correct, we could cheat by re-entering that letter three times, but then, this was just to try out the ideas. A program does what you tell it to do, not necessarily what you wish it to do.

For complicated programs, we usually draw a picture of our thought or decision process. This picture is called a flow chart. For the previous program, I drew first

Now if we test the variables order of precedence, we can rearrange the variables into their natural order by the program.

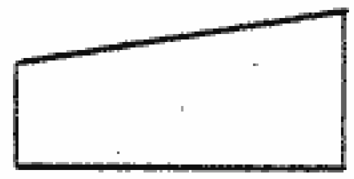
```
10  REM PROGRAM SORT
20  INPUT "FIRST LETTER";FIR$
30  INPUT "SECOND LETTER";SEC$
40  REM EACH LETTER IS INPUT
50  IF FIR$ > SEC$ THEN TEMP$=FIR$:FIR$=SEC$:SEC$=TEMP$
60  REM ALL STATEMENTS ON LINE 50 HAVE CONDITION APPLIED
70  REM REVERSE ORDER ONLY IF NEEDED
80  PRINT "LETTERS ARE";FIR$,SEC$
RUN
```

The variables will be rearranged into their normal ordering. A typical dialog is

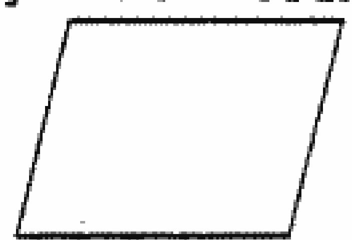
```
FIRST LETTER? M
SECOND LETTER? C
LETTERS ARE C  M
```

This sorting takes advantage of the coding without explicitly using the string commands.

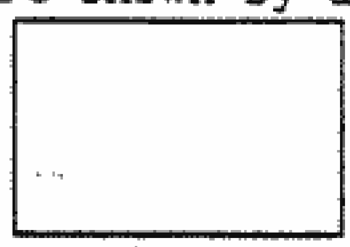
Where we start and end the program and statement numbers in circles as "connection points". Input operations are shown as



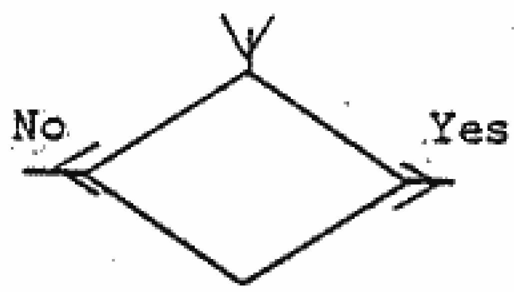
A side view of a keyboard. Printing on the CRT is shown by a



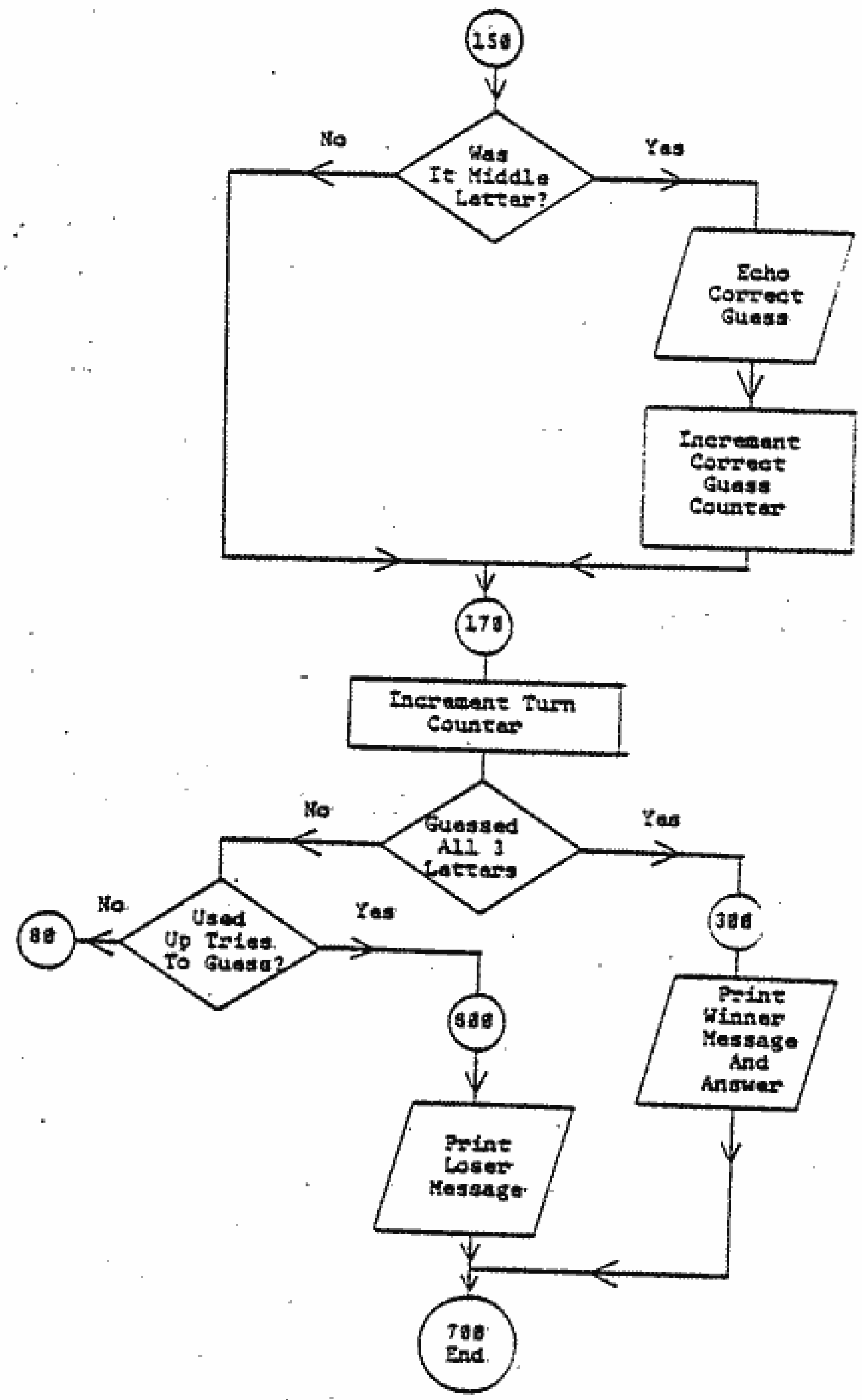
and calculations are shown by a



Branching statements are shown by



where the two possible choices are indicated. These symbols are standard. However, a distinct set of shapes (from any available template) will encourage your use of flow charts. The path of calculations, from one operation to the next, is shown by arrows.

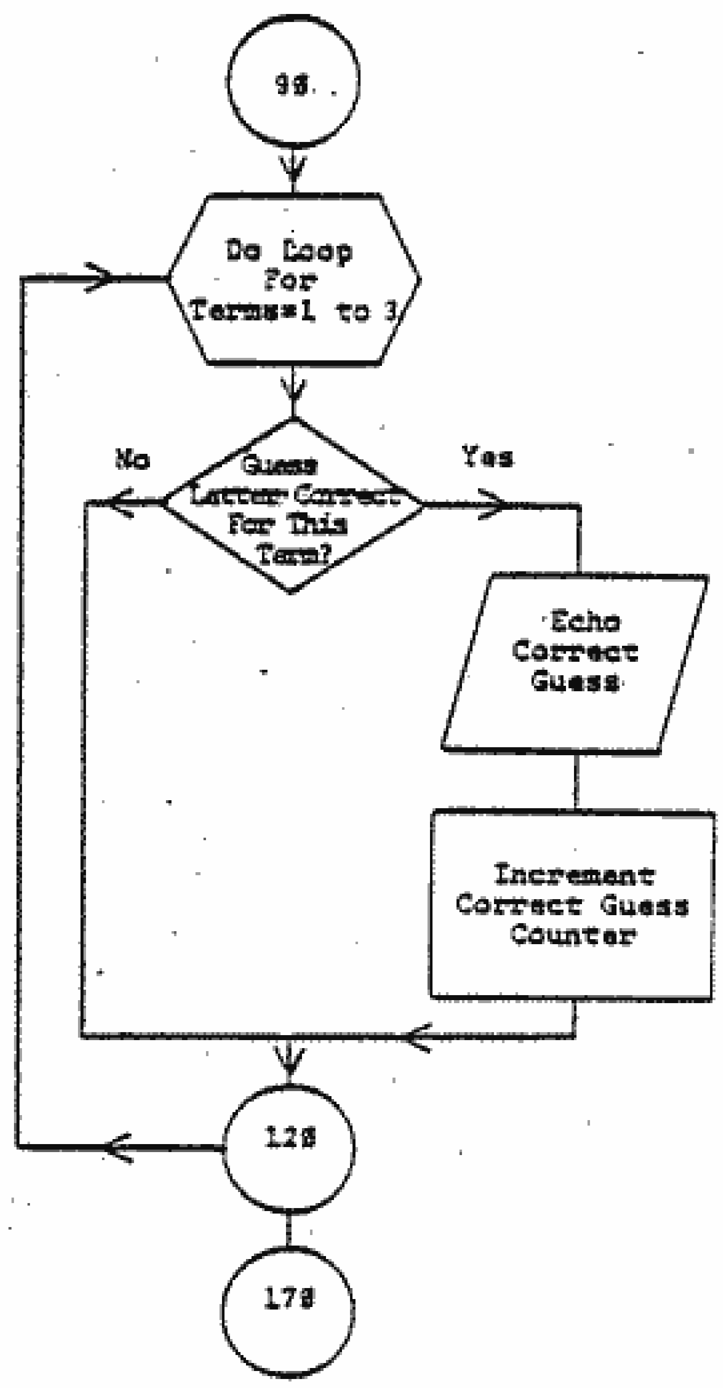


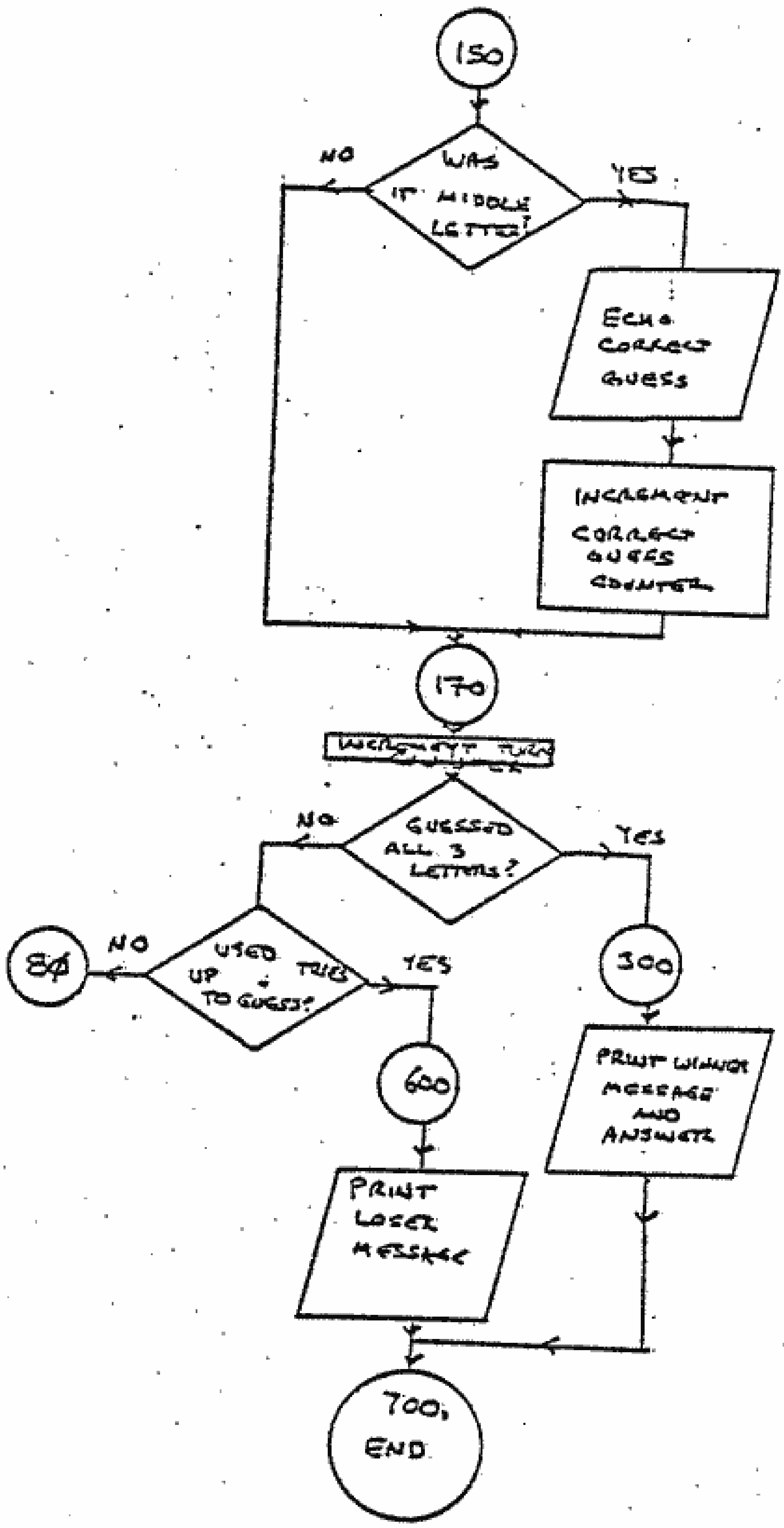
This picture was then directly written as a BASIC program, since the programming decisions had been made.

We can simplify this program by using the MID$ string operation as

```
 90   FOR CHAR=1 TO 3
100   IF MID$ (A$,CHAR,1)=B$ THEN PRINT B$
110   IF MID$ (A$,CHAR,1)=B$ THEN COUNT=COUNT+1
120   NEXT CHAR
130   REM - THE MID$ OPERATION CAN
140   REM - REPLACE THE LEFT$
150   REM - AND RIGHT$ OPERATIONS
160   REM - WITH RESULTING SIMPLICITY
```

The flow chart drawing for this new program segment (statements 90 to 160) can be shown as a loop.





This picture was then directly written as a BASIC program, since the programming decisions had been made.

Each term is considered in the same way, so the loop examines the first, second, and third letters of the answer in order.

If we wanted to rewrite this game program for different length words, this last form would be easier to follow. In your programming, sacrifice anything but clarity.

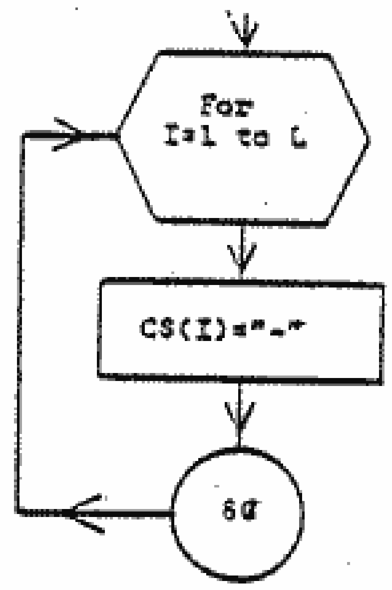
Let's now rewrite the program for words up to five letters in length. We shall output a blank for each letter as a prompt. As the player guesses a correct letter, we'll fill in the blanks and show them (including repeated letters in the word). Most of all, we'll eliminate the chance to cheat by barring reuse of correctly guessed letters, while allowing the opportunity to repeat incorrectly guessed letters.

The former error was a logic error, discovered by playing (testing?) the game. The program writer could have written the program to generously forgive repeated wrong entries, but this would have made the example longer (and easier for the player)!

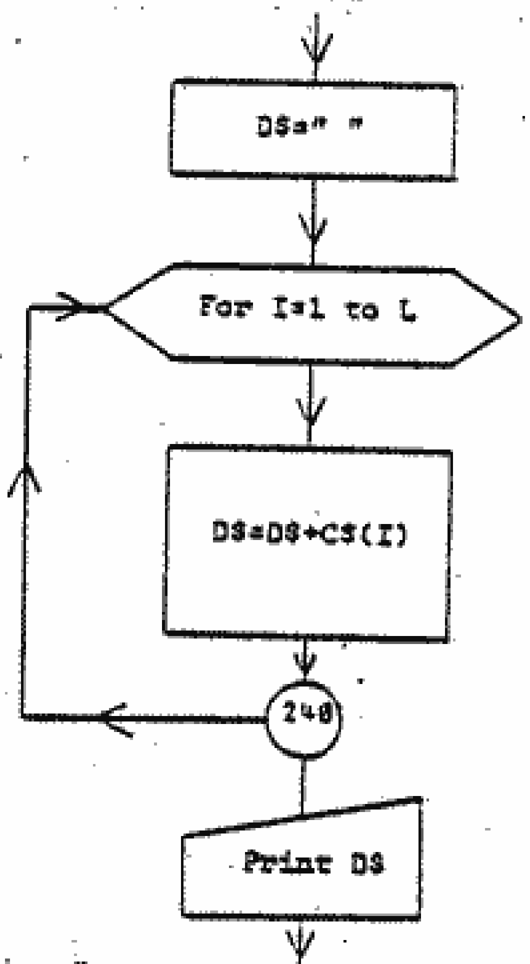
We shall use subscripted variables, such as C$(1), C$(2), C$(3),..., to hold the value of the first, second, third, ..., correctly guessed letter. This will permit clearer printed messages to the player. By using the same variable name, each subscripted variable can be used by merely changing the subscript.

With this more complicated program, we must make a flow chart. We start with an overall flow chart (Figure 4), whose individual boxes get expanded as follows.

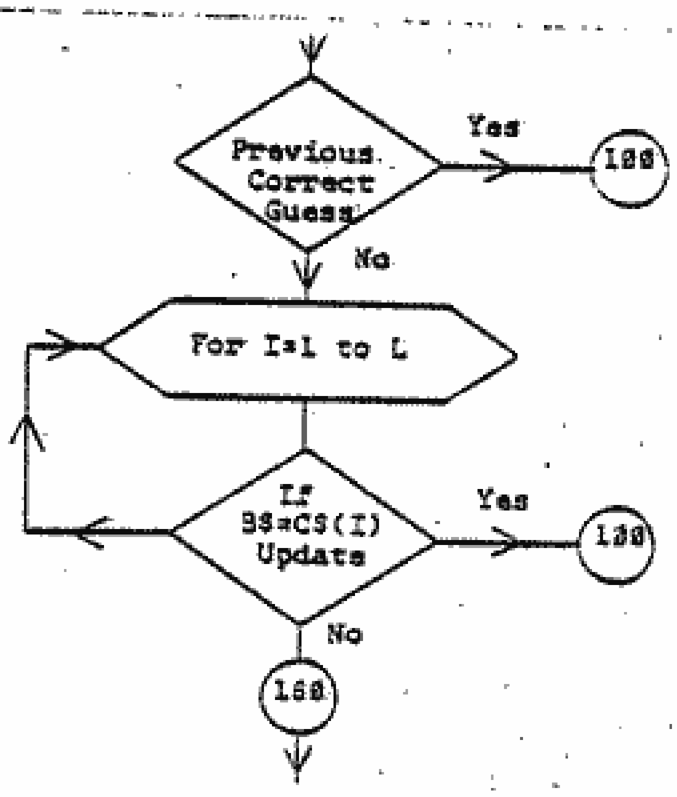
The "clear out answer holder" is expanded as:



and "print present status" becomes



The "previous correct guess" test is:

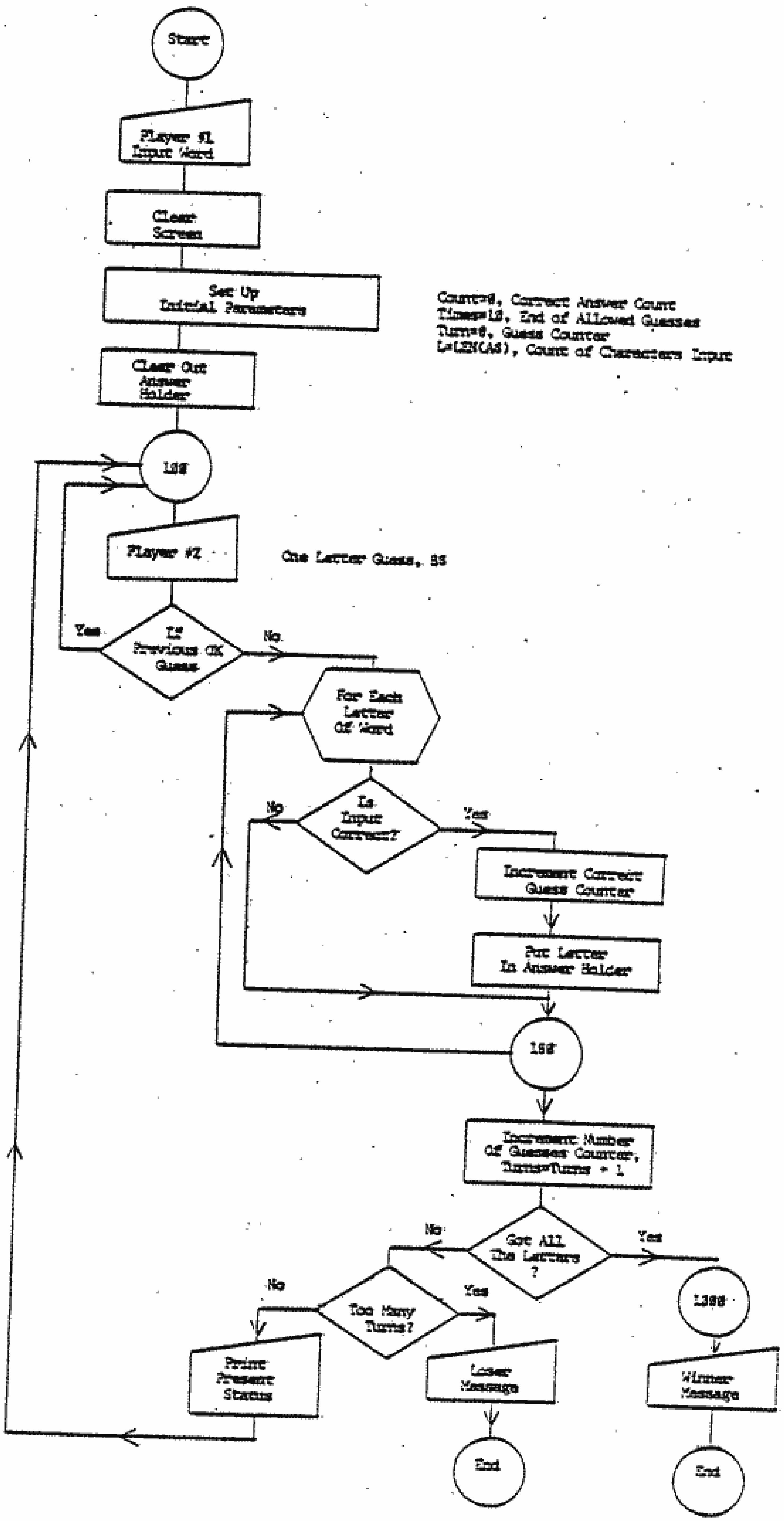


Let's convert these flow charts into a program. If a flow chart is well written, you can code the program as fast as you can type.

```
10   REM PROGRAM:HANG  AUTHOR: L. ROEMER JULY 1979
20   INPUT "PLAYER #1";A$
30   COUNT=0:TIMES=10:TURNS=0:L=LEN(A$)
40   FOR I=1 TO L
50   C$(I)="-"
60   NEXT I
70   FOR I=1 TO 32:PRINT:NEXT I
100  INPUT "YOUR GUESS";B$
110  FOR I=1 TO L:IF B$=C$(I)THEN GOTO 100
120  NEXT I
130  FOR I=1 TO L
140  IF MID$(A$,I,1)=B$ THEN COUNT=COUNT+1:C$(I)=B$
150  NEXT I
160  TURNS=TURNS+1
170  IF COUNT =L THEN GOSUB 1000
180  IF TURNS =TIMES THEN GOSUB 2000
200  D$=""
210  FOR I=1 TO L
220  D$=D$+C$(I)
230  NEXT I:PRINT D$
240  GOTO 100
1000 PRINT"CHEERS"
1100 END
2000 PRINT"BUMMER"
2100 END
```

Note: In Microsoft BASIC, the conditional statement at 150 also imposed the condition on the statement following the colon ":". The colon serves as a separator between BASIC statements which are written on the same line. An equivalent program segment would have been

```
150  IF MID$9A$,I,1)=B$ THEN COUNT=COUNT+1
155  IF MID$(A$,I,1)=B$ THEN C$(I)=B$
```


Start
Player #1 Input Word
Clear Screen
Set Up Initial Parameters
Count=0, Correct Answer Count
Times=10, End of Allowed Guesses
Turn=0, Guess Counter
L=LEN(A$), Count of Characters Input
Clear Out Answer Holder
100
Player #2
One Letter Guess, B$
Yes
Is Previous OK Guess
No
For Each Letter Of Word
No
Is Input Correct?
Yes
Increment Correct Guess Counter
Put Letter In Answer Holder
150
Increment Number Of Guesses Counter, Turns=Turns + 1
No
Got All The Letters ?
Yes
No
Too Many Turns?
Yes
Print Present Status
Loser Message
End
1300
Winner Message
End

The program still could be improved. For example, we have used the variable C$(I) to store the correct guesses. If we want to use more than a ten letter word, additional memory must be reserved for the variable C$(I). This must be done by dimensioning the variable C$(I), for example, for a maximum length of 2Ø letters in a word as

```
5 DIM C$(2Ø)
```

If we do not dimension a subscripted variable, BASIC will default to the assumption of 1Ø subscripts possible. Fortunately, we do not have to dimension the other variables, as they are either single characters or, in the case of A$, a single character string. A character string is a set of characters stored under the single variable name.

When we play this game, the computer user dialog would be, typically,

```
PLAYER #1? GHOST
```

Then after the screen is cleared,

```
YOUR GUESS: G
G____
YOUR GUESS? B
G____
```

This dialog continues until either the winner message of

```
CHEERS
```

or losing message of

```
BUMMER
```

is printed.

Further improvements in the program could be made by providing a preselected vocabulary or having a stick figure drawn as player errors occur. The program works; the style will be your choice.

3. ASCII Code

In using string operations, we must distinguish between a character and its representation inside the computer. For example, to display the number 1, a value of 49 decimal (31 hexadecimal) is sent to the display terminal. This code called ASCII (American Standard Code for Information Interchange), is used for small computer systems. To find the ASCII representation of a character, such as the letter A, we use the BASIC command ASC as follows:

```
10    A$="A"
20    X=ASC(A$)
30    REM THE ASCII REPRESENTATION
40    REM OF THE FIRST CHARACTER IN A$
50    PRINT "THE ASCII CODE FOR";A$;"IS";X
60    END
```

We can turn this process around to find whether 65 is really the code for the letter A by using the command CHR$

```
10    X=65
20    A$=CHR$(X)
30    PRINT "65 CONVERTS TO";A$
40    END
```

One application of the ASCII code conversion is in using POKE's. For example, if the command

```
NEW
```

is to clear prior programs from user memory, you will find the letter "N" in location 741 decimal. To examine this, type

```
PRINT (PEEK(741))
```

which will return

```
78
```

78 is the ASCII code for the letter N. (See appendix for ASCII code list.) Any other symbol in location 741 will disable the command NEW. It would have been easier if we had typed

```
PRINT (CHR$(PEEK(741))
```

Conversion to the expected symbol N would have been done directly.

Another example is found when we change the cursor symbol. The cursor symbol is found in location 9680 decimal. The command

```
POKE 9680,42
```

will make the symbol * into the cursor symbol. We could have used

```
POKE 9680,ASC("*")
```

to achieve the same result, saving looking up the ASCII code. This would be an easier statement to program and a clearer statement to read.

Finally, we can consider some interesting arithmetic. Since the alphabetic characters are ASCII coded sequentially, from 65 decimal for A to 90 for Z, the statement

```
PRINT (ASC("Z")-ASC("A"))
```

will answer

```
25
```

the difference in code of the 26th and 1st characters of the alphabet. Alphabetical sorting can be readily done using this observation.

For example, let's read in two letters, arbitrarily placing the first one in string variable FIR$, the second entry in SEC$.

Now if we test the variables order of precedence, we can rearrange the variables into their natural order by the program.

```
10  REM PROGRAM SORT
20  INPUT "FIRST LETTER";FIR$
30  INPUT "SECOND LETTER";SEC$
40  REM EACH LETTER IS INPUT
50  IF FIR$ > SEC$ THEN TEMP$=FIR$:FIR$=SEC$:SEC$=TEMP$
60  REM ALL STATEMENTS ON LINE 50 HAVE CONDITION APPLIED
70  REM REVERSE ORDER ONLY IF NEEDED
80  PRINT "LETTERS ARE";FIR$,SEC$
RUN
```

The variables will be rearranged into their normal ordering. A typical dialog is

```
FIRST LETTER?  M
SECOND LETTER? C
LETTERS ARE C  M
```

This sorting takes advantage of the coding without explicitly using the string commands.

Chapter III

1. Operating System Organization

An operating system is a program, or set of programs, which supervises the running of your individual programs. That's not a purist definition, but it will do.

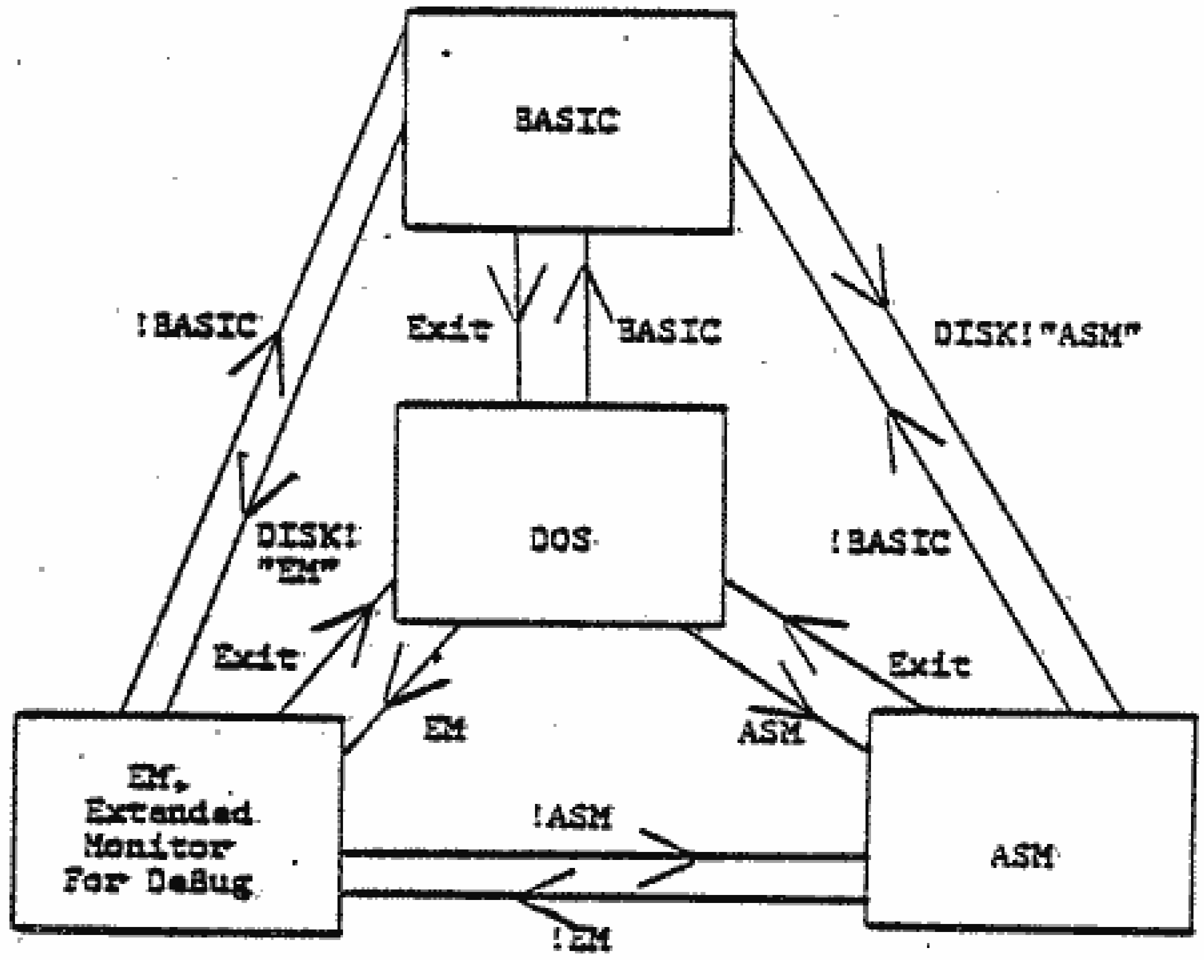
The central part of our disk operating system (DOS on Figure 5) supervises the running of all programs. It can call for three subsidiary (or utility) programs: BASIC, ASSEMBLER language (ASM), and the EXTENDED MONITOR (EM).

BASIC is the program that you will commonly use. It is almost conversational in form. Since it is a high level language, it is very powerful and rapid for program writing.

ASSEMBLER is a shorthand way to write machine language programs. The details are covered in the Ohio Scientific 6500 Assembler/Editor User's Manual and MOS Technology's Microcomputers.

EXTENDED MONITOR provides the ability to inspect, alter, or fill memory locations. It can also move blocks of program from one memory region to another. Details are discussed in the Ohio Scientific Extended Machine Language Monitor User's Manual.

The inter-relation of these programs is shown in Figure 5. The recommended way to go from one program to another is shown beside the direction arrows. These are the commands to be typed.



At boot up time, the operating system will deliver you to the BASIC program as a default.

To illustrate, when you are in BASIC, as shown by the prompt

OK

type

<u>DISK!"EM"</u>

and you will see the EXTENDED MONITOR prompt

:

Upon typing

<u>EXIT</u>

you will leave EM and will be back in DOS as indicated by the * prompter. You can return to BASIC by typing

BA

(Note: valid only if BASIC is still in memory) which brings you back to your starting point.

Since different services are provided by BASIC, EM, and ASM, it is nice to be able to use these programs interchangeably.

2. CREATE a Disk File/DELETE a Disk File

It is useful to be able to name a region of disk for program or data storage. The CREATE utility is set up for this purpose. It reserves room on the disk for your use and enters the file name into the directory for future reference.

To illustrate CREATE, turn your computer on and bring up the disk operating system (OS-65D V3.1). This process is called "booting up" the system. When the BASIC prompt

OK

appears, type

RUN"DIR" RETURN

Respond to the question

LIST ON LINE PRINTER INSTEAD OF DEVICE #2?

by answering

NO RETURN

A listing of your disk directory appears. A typical directory listing follows:

```
OS-65D VERSION 3.Ø
-- DIRECTORY --

FILE NAME     TRACK RANGE
-------------------------
OS65D3        Ø  - 12
BEXEC*        14 - 14
CHANGE        15 - 16
CREATE        17 - 19
√DELETE       2Ø - 2Ø
DIR           21 - 21
DIRSRT        22 - 22
RANLST        23 - 24
RENAME        25 - 25
SECDIR        26 - 26
SEQLST        27 - 28
TRACE         29 - 29
ZERO          3Ø - 31
ASMPL         32 - 32

5Ø ENTRIES FREE OUT OF 64
```

The 1Ø directory files use up 1Ø of the 64 available directory entries. Fifty (5Ø) entries remain free.

If any track between Ø and 39 does not have a file name, we can use that track for our purposes. Let's create a file called SCRTCH. (This is a good idea to have such a file for storing programs during development stages.) File names consist of six or fewer characters; the first character must be a letter. Type

RUN"CREATE" <RETURN>

When asked for a password, respond with

PASS <RETURN>

Then, the computer will respond with

FILE NAME?

You respond with

SCRTCH <RETURN>

The computer response

FIRST TRACK OF FILE?

will be answered with

<u>39</u>

(or whatever track was clear)

Assuming we have only 1 track to copy, the prompt

NUMBER OF TRACKS IN FILE?

is replied with

<u>1</u>

Now when you

<u>RUN"DIR"</u>

you will see this new file "SCRTCH" on the disk.

To DELETE this file, we

<u>RUN"DELETE"</u>

We name the file when

FILE NAME?

appears by typing

<u>SCRTCH</u>

It is common practice to create a scratch file SCRTCH. We can store 2K bytes (approximately 2000 characters) on a track. If we take the memory size in Kbytes and subtract 12K (the approximate system requirements), this leaves your BASIC work space size. For example, a 24K system needs 24K - 12K = 12K bytes of storage. Since 2K bytes fit on a track, your entire BASIC work space could be stored on 6 tracks. Small programs will obviously require far less disk storage.

3. To Write Or Read On Disk

The operating system (OS-65D V3.1) contains simple and powerful routines to handle disk input and output. These routines permit using low cost disk storage rather than using the more expensive random access memory (RAM).

I. A simple connection for storing BASIC programs is available.

If we have already created a file, say "SCRTCH" (see preceding section), then a simple program such as

```
10  PRINT "NEW TEST"
20  END
```

can be stored on the file "SCRTCH" by typing

DISK!"PUT SCRTCH" <RETURN>

If you now type

NEW <RETURN>

LIST <RETURN>

nothing will be printed, since the work space was cleaned by the NEW command.

To load the program from disk into your BASIC work space, type

DISK!"LOAD SCRTCH" <RETURN>

Then the LIST command

LIST <RETURN> → false — Lists the DIR program write VEG overlayed.

will result in the listing of the previously stored program.

II. Another method to store and retrieve the program on SCRTCH is available. You could have exited BASIC by typing

EXIT <RETURN>

Then respond to the DOS prompt

A*

by typing

<u>PUT SCRTCH</u> <RETURN>

to store the program directly under control of DOS.

The copying of file "SCRTCH" into the work space is accomplished by typing

<u>LOAD SCRTCH</u> <RETURN>

III. If you wish to be able to specify the disk locations and memory locations yourself, a more detailed set of commands are CALL and SAVE.

These commands are used after the operating system prompt

A*

as

CALL address = track, sector <RETURN>

and

SAVE track, sector = address/page <RETURN>

These commands transfer a specified track (1 to 39), sector (1 to the maximum you have used on that track). A page is 256 bytes. Each sector is an integer multiple of pages, i.e., 1, 2, 3 pages of 256 bytes each. The address must always be a four digit hexadecimal value, track must be two decimal digits (so track 2 is written Ø2), and sector is one decimal digit. Pages must be one hexadecimal digit within the range 1 to 8. A given sector can be referenced only if all lower numbered sectors exist on the specified track.

The CALL and SAVE commands are particularly suited to storing and retrieving machine code programs. An example of this is shown in the use of disk copy routines given in the appendix. The CALL and SAVE also permit storing data on a track without the requirement of creating a named file.

Since all these routines can be invoked within a BASIC program, we have the ability to run complete BASIC programs which use other BASIC and machine code programs, brought in as needed from disk. This provides the ability to use large programs, small parts of which are brought into memory as needed.

However, you will often want to use these routines, CALL and SAVE, under BASIC. The DISK! command can be used to gain access to the operating system commands while remaining in your BASIC program. For example, to SAVE a program on track 39 for 1 sector, where the program is resident at memory location 3279 hexadecimal, and it is less than one page (256 characters) long, we could use

DISK!"SAVE 39,1 = 3279/1" <RETURN>

Likewise, to recall this same program back into these same memory locations, we write

DISK!"CALL 3279 = 39,1"<RETURN>

Caution is urged, as it is possible to bring your disk program on top of a program you are using. This will destroy the program which is overlayed. Each command that gives you additional power or discretion carries the need for additional caution.

Chapter IV

Disk Utility Programs

Some housekeeping programs are available on your system disk. We have already looked at the use of CREATE and DIR. When we have no further need of a disk file, or when we need to make room on your disk for a new file, we have a need of the DELETE utility.

1. DELETE Utility

The DELETE utility is invoked by

RUN"DELETE" <RETURN>

As in any utility where you run the risk of deleting valuable programs or data, the utility program requires

PASSWORD?

to which you respond

PASS <RETURN>

The utility then requests the name of the file to be deleted as

FILE NAME?

to which you name the file name to be deleted. Upon deletion, the file name will be missing from the directory. When a file is DELETEd, only the name is removed. The program or data which resided on disk will still be present. If we wish to erase the data which is present in a file, we invoke the ZERO utility.

2. ZERO Utility

The ZERO utility will fill a named file with zeros. This is valuable when we wish to place a background of zeros on which we can readily recognize our data or program. We invoke the ZERO utility by

RUN"ZERO" <RETURN>

When the utility requests

PASSWORD?

Again, we use the password

PASS <RETURN>

The utility then requests the name of the file to be zeroed as

FILE NAME?

Reply with the file name. When the question

IS IT A NORMAL 8 PAGE DATA FILE?

is printed, you answer either YES or NO. Usually, you answer YES. If you answer NO, the following message is printed

THEN HOW MANY PAGES PER TRACK?

(Each page contains 256 bytes.) You may reply to the question with any number from 1 to 8.

When the file has been ZEROed, the utility will return you to the BASIC program.

3. RENAME Utility

For convenience, we sometimes wish to change file names. The directory entry for file name can be changed by

RUN"RENAME" <RETURN>

The utility requests your

OLD NAME?

You respond with the existing file name which you want changed. The program responds

RENAME OLD NAME TO?

You type the new file name as your response. File names may be 1 to 6 characters, with the first character a letter.

Upon completion of the RENAME utility, the user is returned to BASIC.

4. DIRSRT, Directory Sort Utility

As we add entries to the disk directory, these are added in order of entry. As the directory gets full, we will want the directory to be placed in order. The choices are alphabetical order of file name or numeric order of track. This utility is invoked by

RUN"DIRSRT" <RETURN>

You will be asked

SORTED BY NAME OR TRACK (N/T)?

Reply N or T to select alphabetical or numerical order.

When asked

LIST ON LINE PRINTER INSTEAD OF DEVICE #2?

answer

NO <RETURN>

to avoid a printer listing (unless you have a printer on Device #4 port and want a listing).

If you enter an unsatisfactory answer, the utility will not sort the directory.

5. SECDIR, the Sector Directory Utility

The SECDIR sector directory utility will list the track contents by sector. A sector is a subdivision of the track, which is divided into pages of 256 bytes each (for a maximum of eight pages per track). Since the operating system can load sectors, without having to load an entire track, this utility provides a check on disk utilization. To invoke its use,

RUN"SECDIR" <RETURN>

The utility will inquire

FIRST TRACK?

respond with any track number between 1 and 39. To the question

LAST TRACK?

you have the same choice of answers (1 to 39). If we had examined track 13 (which contains the COPY utility, typically) we would give the first and last track as 13, where upon the listing would appear

TRACK 13

Ø1-ØS

indicating that track 13 contained one sector of five pages.

6. The TRACE Utility

The TRACE utility will display the line number of each statement prior to execution. This utility will permit seeing the sequence of calculations in a BASIC program. The TRACE utility is invoked by

RUN"TRACE" <RETURN>

The TRACE utility will respond

ENABLE OR DISABLE (E/D)?

To invoke the utility type

E <RETURN>

If you had already invoked the TRACE, you can turn it off by invoking the utility and responding

D <RETURN>

as the chosen response.

The first number the computer types is

160

the last line of the utility program. The TRACE utility doesn't affect the sequence of operation of the program.

7. Random Access File List Utility, RANLST

This utility program may be used to list the contents of a random access file either a single record at a time or in groups of contiguous records. (Random access files are labeled with a record number in contrast to sequential access files.) The program assumes 128 byte records. To list a random file, type

RUN"RANLST" <RETURN>

The program output and the kind of input you may enter in response are as shown below. Any unacceptable response will result in an error message and/or a repeat of the request for input.

RANDOM ACCESS FILE READ

FILE NAME?

Enter the name of the random access file to be listed.

EXAMINE SINGLE RECORDS OR GROUPS (S/G)?

Enter S or G. If S is entered, the number of the single record to be listed is requested.

RECORD NUMBER?

Enter the number of the record to be listed. (Records are numbered from zero through n.) The specified record is listed, then the RECORD NUMBER question is again asked. To terminate the program, merely type a <RETURN> to this question.

If G is entered above, the range of record numbers to be listed are requested.

FIRST RECORD?

Enter the number of the first record to be listed.

LAST RECORD?

Enter the number of the last record to be listed.

The specified records are listed, then the "SINGLE RECORDS OR GROUPS" question is again asked. To terminate the program, merely type a <RETURN> to this question.

Note that this program reads and lists a single string from the start of each record. Random files with more than one entry (an entry is a string of printing characters followed by a return) per record will not be fully listed by this program.

8. <u>Sequential File Lister Utility, SEQLST</u>

This utility program may be used to list the contents of a sequential file. A sequential file is one in which all entries within the file are contiguous with no intervening gaps. To list a sequential file, type

<u>RUN"SEQLST"</u> <RETURN>

The program output and the kind of input you may enter in response are as shown below. Any unacceptable response will result in an error message and/or a repeat of the request for input.

SEQUENTIAL FILE LISTER

TYPE A CONTROL C TO STOP

FILE NAME?

Enter the name of the sequential file to be listed.

The specified file is listed until you type a CONTROL C or the end of the file is reached in which case the program terminates with the following end-of-file message:

ERR #D ERROR IN LINE 100

OK

9. <u>CHANGE, the Utility for Work Space and Input/Output Change</u>

The CHANGE utility services Input/Output parameter changes. The normal (default) value for printer width is 132 spaces. These are the printable characters, which get padded by blanks at output. Carriage return and line feed are automatically added beyond these 132 spaces. Additionally, the number of printer fields (the number of variables which can be printed across a page) has a default value of 8, one less than the number of whole 14 character columns that will fit within 132 printable characters. Any change in printer width will change the number of printer fields accordingly.

To invoke the CHANGE utility, type

<u>RUN"CHANGE"</u> <RETURN>

The program output and the kind of input you may enter in response are as shown below. Any unacceptable response will result in an error message and/or a repeat of the request for input.

CHANGE PARAMETER UTILITY

THE TERMINAL WIDTH IS SET FOR 132

DO YOU WANT TO CHANGE IT (Y/N)?

Enter YES or NO. If you enter YES, the program requests a new value for the terminal width.

NEW VALUE?

Enter a new value from 14 through 255.

The next option to change is available memory. Since you will receive a default value of the maximum memory available, any change will reduce the memory available for BASIC or ASSEMBLER use. By denying memory allocation to BASIC and ASSEMBLER, room may be reserved for machine language programs.

The CHANGE utility, after the prior Input/Output changes, will reply:

BASIC & ASSEMBLER USE xx K WORK SPACES (yyy PAGES)

WOULD YOU LIKE TO CHANGE THIS (Y/N)?

The work space is the main memory available to the system software. Each K (1Ø24 bytes) contains four 256 byte pages. A change to this parameter will make a portion of highest memory unavailable to systems software. Note that such memory will not be included within LOAD/PUT files.

Enter YES or NO. If you enter YES, the program requests the number of pages to be used by system software.

HOW MANY PAGES SHOULD THEY USE?

Enter a number of pages from 5Ø through 191.

The program contines with:

CHANGE BASIC'S WORK SPACE LIMITS (Y/N)?

Enter YES or NO. If you enter NO, the program terminates. If you enter YES, the program requests the following:

HOW MANY 8 PAGE BUFFERS DO YOU WANT BEFORE THE WORK SPACE?

Enter Ø, 1 or 2 to reserve that many track buffers at the beginning of the work space. Note that device 6 memory buffered I/O uses the first buffer by default while device 7 uses the second buffer by default. Of course, these defaults can be changed with appropriate POKEs. If no buffers are specified, the program asks:

WANT TO LEAVE ANY ROOM BEFORE THE WORK SPACE?

Enter YES or NO. If you enter NO, the program outputs the address of the start of the BASIC work space as shown below. If YES is entered, proceed to the "HOW MANY BYTES?" question below.

If one or more buffers was specified, the program continues with:

WANT TO LEAVE ANY ADDITIONAL ROOM?

Enter YES or NO. If you enter YES, the following question is asked:

HOW MANY BYTES?

Enter the number of additional bytes to be allocated before the start of the work space.

The program then outputs the new address for the start of the work space and the total number of bytes reserved for buffers, etc.

THE BASIC WORK SPACE WILL BE SET TO START AT aaaaa

LEAVING bbbb BYTES FREE IN FRONT OF THE WORK SPACE

IS THAT ALRIGHT?

Enter YES or NO. If you enter NO, the program requests that you specify an exact lower limit address for the work space.

NEW LOWER LIMIT?

Enter a lower limit address. The program then confirms this value by outputting:

bbbb BYTES WILL BE FREE BEFORE THE WORK SPACE

The program then continues with:

YOU HAVE xx K OF RAM

DO YOU WANT TO LEAVE ANY ROOM AT THE TOP?

Enter YES or NO. If you enter YES, the following question is asked:

HOW MANY BYTES?

Enter the number of bytes of Random Access Memory (RAM) to be allocated between the top of the work space and the end of main memory. The program then outputs:

THE BASIC WORK SPACE WILL BE SET TO END AT ccccc

LEAVING dddd BYTES FREE AFTER THE WORK SPACE

IS THAT ALRIGHT?

Enter YES or NO. If you enter NO, the program requests that you specify an exact number limit address for the work space.

NEW UPPER LIMIT?

Enter an upper limit address. The program then confirms this value by outputting:

eeee BYTES WILL BE FREE AFTER THE WORK SPACE.

Note that the reservation of space after the work space is not recorded on disk with a program when it is saved in a file. The allocation is only recorded as a RAM resident change to the BASIC interpreter and remains in effect until explicitly changed again, or BASIC is reloaded by typing BAS in the DOS command mode. Later, running a program that results in an "Out of Memory" (OM) error may be the result of a reduced work space that is no longer required. Program output continues with:

YOU WILL HAVE fffff BYTES FREE IN THE WORK SPACE

IS THAT ALRIGHT?

Enter YES or NO. If NO is entered, the Change Parameter Utility Program restarts from the beginning. Otherwise, the requested changes are made, the work space contents are cleared and the program terminates.

Chapter V

Peripherals, An Overview

A computer's value over a calculator depends on its ability to change its sequence of computation based on the results already computed. This is particularly important when the values used in computation (decision) are data from or to external devices. External devices use the data, binary 1's and Ø's, sent over lines from the computer as output. The 1's and Ø's are represented by nominal 5 volt and Ø volt levels (TTL logic levels), respectively. Likewise, external devices can send data as input to the computer. Again, standard TTL logic levels are used.

Control, in the C4P, includes being able to turn on/off (and set the level of) AC controlled devices, such as lamps, motors, and appliances. Control also includes being able to supervise security alarms, as well as numerous status switches. All of these capabilities allow device operation while the computer is doing tasks of a more immediate priority.

In the next two sections, "Appliance Control" and "The Home Security Alarms", the most popular applications are considered. Then, in greater detail, the many possibilities of additional options and capabilities are considered. By combining the capabilities of several features in one program, great flexibility and power can be obtained. All of this is controlled by your readily written BASIC program, based on the examples that follow.

1. Appliance Control

Without running any wires your C4P can operate lamps and small appliances when equipped with the AC-12 options! This is accomplished by the BSR X-10(C) a remote AC signaling system. The computer activates the BSR command console which, in turn, sends a signal over the existing home wiring. This signal is sensed at the appropriate device by a small switch module plugged into the AC outlet. The switches are modules which plug into the wall sockets (110 volt AC power lines). The appliances are plugged into these modules.

Two types of switches are available, a lamp switch and an appliance switch. A continuously dimmable lamp switch provides adjustable incandescent lighting levels (up to 300 watts per lamp) throughout a building. A relay actuated (on-off) appliance switch provides control of larger devices such as lamps (up to 500 watts), motors (up to 1/3 HP), or current loads of up to 15 amperes.

Each remote switch module has two dials. One selects "house code". You have sixteen choices indicated by the red letters A through P. The "house code" on the remote module must match the "house code" on the control console. The various "house codes" prevent signals from other computers from actuating your remote switch modules. Each switch module also has a "unit code" dial (up to 16 units can be addressed), which permits great flexibility in home/office control.

Lights in each room can be put on a different module. Computer control permits turning lights on and off, one room at a time.

The timing and sequence, following your directions under computer control, can be specified with simple commands.

In order to run your own AC control programs, we need to borrow support programs from your system disk (OS-65U V3.1 HC).

Software control of these remote switches requires running the previously stored program, "AC", by typing

RUN"AC"

This brings the device driver programs from disk. The device drivers permit a relatively simple set of commands to control the more complex functions of the lamp and appliance switch modules. Your user program must contain

a) A POKE to set the display screen state

POKE 249,1

will set a 64 by 32 character B&W (sound off) display in the same manner as you can use

POKE 56832,1

to set the display state (discussed in video section)

b) Address 548 (224 hex) and 549 (225 hex) must contain the low and high bytes of the address of the AC driver routines.

These are set for us by the command

POKE 548,127

POKE 549,5Ø

Having taken care of the three required POKEs, we can now write device driver programs.

The AC driver routines utilize a new BASIC command, ACTL, with the following format

ACTL DEVICE,COMMAND

where DEVICEs are numbered 1 to 16 and the COMMAND choices are as follows:

Function	COMMAND
Turn on device	65
Increase brightness (lamps only)	66
Turn all lights on (lamps only)	67
Turn off device	68
Decrease (dim) brightness (lamps only)	69
Turn all devices off	7Ø

(The total range of dimming (brightening) is accomplished in 12 steps.)

If a light is in the off state, brightening it will result in it being turned on, first.

This ACTL command can be used to turn on device number 4 (plugged into a module which has had its unit dial set to 4) by

ACTL 4,65 <RETURN>

Multiple devices, for example numbers 4 and 5 can be turned off, using the format

ACTL DEVICE1,DEVICE2,...,COMMAND <RETURN>

as

ACTL 4,5,65 <RETURN>

Similarly, use of the format

ACTL DEVICE,COMMAND,COMMAND,...,COMMAND <RETURN>

permits brightening device 4 through 3 of the 12 levels of brightness by

ACTL 4,66,66,66 <RETURN>

Another variation of the ACTL command is

```
ACTL DEVICE1
ACTL DEVICE2
     :
ACTL COMMAND
ACTL COMMAND
     :
```

which can be used to slowly brighten device 1 and 2 simultaneously by

```
100  ACTL1
200  ACTL2
300  FOR TIME=1 TO 12
400  FOR DELAY=1 TO 100 : NEXT DELAY
500  ACTL 66
600  NEXT TIME
```

For safety considerations, the command for "all off" (70), which turns off all lamps and appliances, was not matched with an "all on" command. The "all lights on" affects only the lamp modules.

We now have software commands to control one of the peripheral devices on the C4P system. New additions to the peripheral family will be serviced in a similar manner to the devices already described. When we have examined each of the available devices, we shall put the devices together in a REAL TIME system.

2. The Home Security Alarms

The first level of home security can be met with the home security alarms alone. These devices provide checking for fire, intruders or tampering with your vehicles. All alarms report their status by radio-control to the home control module, connected to your computer (on J3 of the C4P back panel, Figure 1B). Each alarm module contains the sensor, battery power, and a radio transmitter to assure a reliable and tamper resistant operation.

The fire alarm can sense temperature (thermal contact) or smoke (ionization detector). The intruder alarms are silent magnetically actuated door or window position sensors. By combining these alarms with computerized response, such as automatic dialing of the telephone emergency numbers, a rapid response to critical situations can be managed. The car alarm senses car battery voltage change; a door opening or the radio or lights left on would actuate the alarms. The intrusion and car alarms permit choice of immediate alarm or delaying for 15 seconds prior to actuating (sounding) the alarm. This gives time for you to disable the alarm when you enter the house normally.

Additionally a hand held alarm is available for handicapped or bedridden persons. All alarms have an effective radius of 200 ft. (60 meters) from the alarm site to the computer home control module.

The alarms are located at the computer address 63232 and the alarm control at 63233. The alarms are enabled (permitted to report back to the computer) by setting locations 63233 and 63234 to the values given in the program below:

```
10  REM PROGRAM AID ; LISTEN FOR HOME SECURITY ALARMS
20  ENABLE=0 : HEAR=0 : TRIP=0
30  ALARM=63232 : CTRL=63233 : START=4
40  POKE CTRL, ENABLE : POKE ALARM, HEAR : POKE CTRL, START
50  REM SET UP TO LISTEN TO ALARM LINES
60  FIRE=1 : BURGLER=2 : CAR=4 : MISC=8
70  TI=PEEK(ALARM) AND FIRE
80  T2=PEEK(ALARM) AND BURGLER
90  T3=PEEK(ALARM AND CAR
100 T4=PEEK(ALARM) AND MISC
110 REM TESTS T1,T2,T3, AND T4 TO CHECK IF ALARM TRIP
120 IF T1=TRIP THEN PRINT "FIRE"
130 IF T2=TRIP THEN PRINT " BURGLER"
140 IF T3=TRIP THEN PRINT "CHECK CAR"
150 IF T$=TRIP THEN PRINT "MISC ALARM"
160 GOTO 70
170 END
```

In later examples, we shall incorporate further alarm response. Your alarm monitoring can be done while other programs are being run. This powerful technique is available when you use the real time monitor, RTMON. Many computer controlled responses can also be called. For example, AC, Appliance Control can regulate light levels or sound warnings; automatic telephone dialing can summon aid.

The user has the ability to maintain detailed supervision of home security with the simplicity of conversational instructions in BASIC.

Chapter VI

General Peripherals, Their Use

The distinguishing characteristic of the home security/control features is the ability to control external devices by use of your C4P computer. But the C4P is still a "personal" computer.

The traditional devices, such as a line printer or modem, can be attached (through standard connectors) to your C4P system ("modulator-demodulator", used to connect the telephone to the computer). The computer signals the modem to generate or receive tones which can be carried by the telephone lines. A line printer provides convenient data logging for record keeping in the home or business. Further, the line printer provides a documentation aid in program development. For these reasons, a line printer is often an early choice in expanding a user's system. More recently, low cost modem circuits have permitted connecting the user's C4P to a telephone interface. This allows conversations between computers. In many environments, students and others can find access to the large mainframe computers at their school or place of employment which allow dialing-in for off-site use. Your modest investment in a C4P system can unlock all the facilities of a large computer center at your convenience, without travel from your chosen location.

The home or business C4P system will also have need of checking switch positions for open or closed status. Home security systems provide an obvious application. We'll use a greenhouse temperature control and alarm system in a later example. Fire alarms, counting events, timing occurrences, regulating intervals, signaling by tone generation, and even voice generation are possible with your C4P system.

The demonstration disks have shown you part of the power of your system. The applications which you will write can bring this power to tasks of your choosing.

Let's look at the characteristics and use of the external devices. The examples which follow will show

1) device assignments, including use of printer, modem and disks

2) tone and music generation

3) joystick and keypad entry of data. (Also, these input devices provide convenient game control as a bonus.)

4) voice generation for communications

and

5) control using timers and a real time monitor to supervise the tasks above.

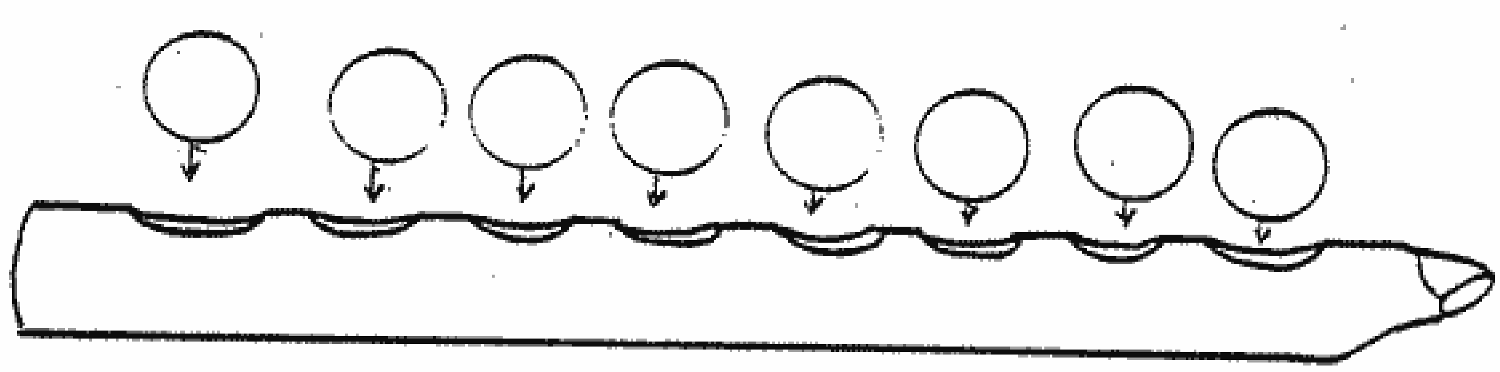
Now, let's examine those applications.

1. The Printer, Modem and Other Input/Output Devices

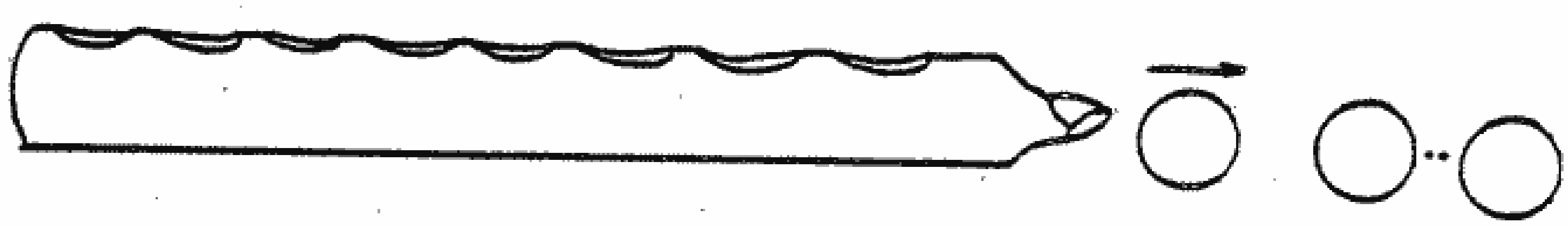
Each character which we store or move is represented by 8 bits (ones or zeros). Normally, we have data on eight data lines (called a data bus), simultaneously. This is convenient when the cost of maintaining multiple lines is low, due to short line lengths. For longer lines, extra circuits for each line are necessary to maintain data signal fidelity. Also, the cost of maintaining long data lines must be balanced against the speed and convenience of having all data bits simultaneously available.

Certain devices require serial data handling. Serial data handling treats one bit (off-on) at a time, rather than all data bits simultaneously. The serial devices are low speed, with no ability to simultaneously transmit or receive more than one bit at a time. Bits are collected by the serial data device until a complete character is available. Then, when the complete character has been received, it is sent in parallel (all bits simultaneously) to the computer for processing. Serial data is handled by an Asynchronous Communications Interface Adapter (ACIA) which converts the parallel (simultaneous) data into serial data for transmission (or reverses the process for reception).

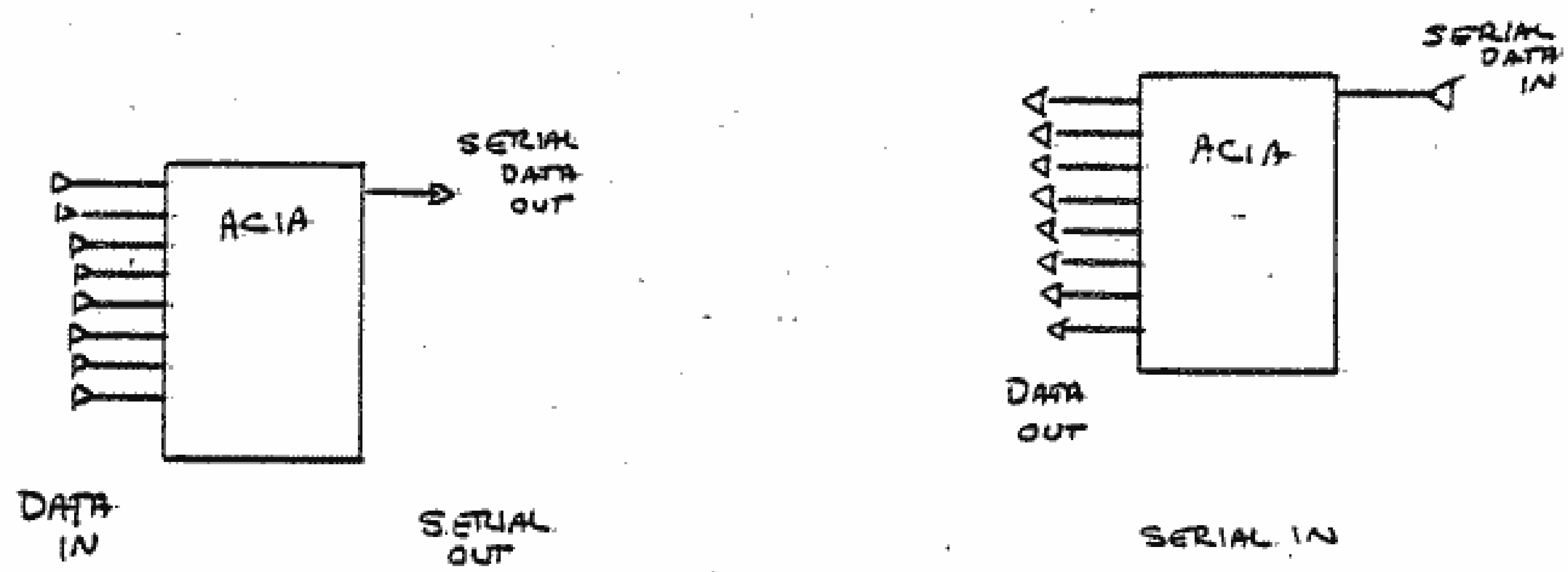
A simple analog might suggest the function of the ACIA. Consider that the input from a computer is typically 8 parallel, simultaneous, input bits. We can represent the function of the ACIA by a child's whistle, where we input data (peas) in parallel



As data is output (removed from the ACIA device) we represent the sequential or serial flow as



Data flow in both directions could be accommodated (though not simultaneously) by reversing the process. The electrical equivalent of the whistle analog would be



where control and timing for the ACIA must be provided, additionally.

This serial (or sequential) handling of bits requires fewer wires for data transmission, but the data handling rates are consequently reduced. This is no disadvantage if the device to which we sent data is limited to low mechanical speeds (such as printers, plotters) or low data rates (telephone lines and their modems).

The system will normally be set up with the information handling rate (baud rate) set at 1200 bits per second (1200 baud). For the modem use, this must be changed to 300 baud. The two choices are given by

```
POKE 64512,1 : REM 1200 BAUD RATE
```

or

```
POKE 64512,2 : REM 300 BAUD RATE
```

In contrast to the ACIA, Parallel Interface Adapters (PIA's) handle all 8 bits of a character's data simultaneously. These serve as interface to the outside (of the computer) world.

2. <u>I/O Distribution</u>

The simplest way to send data to the ACIA is to inform the Disk Operating System (DOS) that the ACIA is to be an output port. The command, responding to the DOS prompt

```
A*
```

is

```
IO ,01
```

This assigns the ACIA as the sole system output port.

The general form of I/O distribution is

IO nn	to assign input devices only
IO ,mm	to assign output devices only
IO nn,mm	to assign both input and output devices

Note that these numbers, nn, mm, are in hexadecimal (base 16). Each device number assignment must be a two digit number selected from the following list:

Hex nn	Input Device Code	Hex mm	Output Device Code
ØØ	Null	ØØ	Null
Ø1	Serial Port (ACIA at FCØØ)	Ø1	Serial Port (ACIA at FCØØ)
Ø2	Keyboard on 44Ø/54Ø Board	Ø2	Video on 44Ø/54Ø Board
Ø4	UART on 43Ø Board	Ø4	UART on 43Ø Board
Ø8	Null	Ø8	Line Printer
1Ø	Memory	1Ø	Memory
2Ø	Disk Buffer 1	2Ø	Disk Buffer 1
4Ø	Disk Buffer 2	4Ø	Disk Buffer 2
8Ø	55Ø Board Serial Port	8Ø	55Ø Board Serial Port

Each of the device codes listed is a hexadecimal value corresponding to one bit or device. For example, the ACIA (device Ø1) is given by bits

```
ØØØØ ØØØ1
```

and the video board (CRT terminal) is device Ø2, given by bits

```
ØØØØ ØØ1Ø
```

We can use both devices simultaneously by specifying the device with a bit pattern

```
ØØØØ ØØ11
```

which is hexadecimal Ø3. Therefore

```
IO ,Ø3
```

will send data to the CRT terminal and the device on the ACIA port, simultaneously. Multiple output devices may be used (in contrast to only single input devices). We could have attached either a serial printer or a telephone line modem to the ACIA output as the external device. However, only one device, the modem or the printer, may be attached at any one time. That is, you can't have power applied to the printer and modem simultaneously. You may store modem data on disk files for later printing, so this is not a difficult restriction. Only one device will have its input accepted at one time. Since the I/O command may specify multiple input devices, a priority rule is established. On input, the lowest numbered devices gets to talk. Other devices are ignored. This gives the modem port high priority.

As an alternative to the I/O command, the ACIA port may be addressed by using the ACIA control register address of FCØØ hexadecimal (64512 decimal) and its data register of FCØ1 hexadecimal (64513 decimal). Reading or writing can be accomplished using the BASIC PEEK and POKE commands.

The simple program

```
  5 REM PRINTER PROGRAM
 1Ø POKE 64512,1 : REM SET 12ØØ BAUD RATE
 2Ø A$="NOW IS THE TIME FOR ALL GOOD MEN"
 3Ø FOR T=1 TO 2Ø : REM PRINT 2Ø TIMES
 4Ø FOR K=1 TO LEN (A$)
 5Ø A=ASC(MID$(A$,K,1))
 6Ø FOR DELAY=1 TO 2 : NEXT DELAY
 7Ø REM WE HAD A SLOW PRINTER
 8Ø POKE 64513,A
 9Ø NEXT K : REM MESSAGE COMPLETE
1ØØ POKE 64513,1Ø : REM LINE FEED PAPER
11Ø POKE 64513,13 : REM CARRIAGE RETURN
12Ø NEXT T : REM DO ALL 2Ø LINES
13Ø END
```

prints the message

NOW IS THE TIME FOR ALL GOOD MEN

twenty times, illustrating the ACIA function.

This method is less convenient than the I/O command discussed previously and should be used only to overcome new device limitations (such as the need for the additional delay created by a delay loop in line 60).

3. Other Devices

For other devices, it is probably easier to accept the device handlers built into the BASIC programs. Under BASIC, the devices are numbered sequantially, 1 to 9. This renumbering is distinct from the previous I/O command example. Under BASIC, the devices which are available are

Device Number	Input Devices	Device Number	Output Devices
1	Serial Port (ACIA	1	Serial Port (ACIA)
2	Keyboard on 440/540 Board	2	Video on 440/540 Board
3	UART on 430 Board	3	UART on 430 Board
4	Null	4	Line Printer
5	Memory	5	Memory
6	Disk Buffer 1	6	Disk Buffer 1
7	Disk Buffer 2	7	Disk Buffer 2
8	550 Board Serial Port	8	550 Board Serial Port
9	Null	9	Null

The DOS I/O command previously discussed remains in effect until it is reset or an error occurs. If an error occurs, the default value is set (start up value). In contrast, the device numbers above can be assigned for each input/output operation as needed. For devices other than those set up by the DOS I/O command, we could use the device assignments immediately above.

For example, to read from the keyboard and write on the printer attached to the ACIA, we may use

```
10  INPUT #2,A$ : REM KEYBOARD INPUT
20  PRINT #1,A$ : REM TO PRINTER ON ACIA
30  LIST #1     : REM AND LIST PROGRAM, TOO
RUN
```

We will get the input prompt

```
?
```

After typing a message (72 characters or less) and a <RETURN> , the message and the program will be printed on the serial printer.

4. Disk Use

As an input/output device, disks can be used in a similar manner.

However, prior to using the disk, the user should provide for protecting his buffer areas by running the CHANGE program as

```
RUN"CHANGE"
```

You should respond to the terminal width change with

NO <RETURN>

and respond to a request to change the BASIC and ASSEMBLER use of memory by

NO <RETURN>

but respond to the work space limit change by

YES <RETURN>

The CHANGE program will ask you "how many 8-page buffers before the work space." (Remember each page contains 256 characters.)

There are only two valid responses here (1 and 2)

1 if only one file is to be used

2 only two files must be open simultaneously

For the example that follows, 1 is sufficient. No additional room is required, so respond

<u>NO</u> <RETURN>

to that question. You also need not request any room at the top for this example.

The small differences between a disk and other devices are the need to open a disk file by name as

```
DISK OPEN,6, "FILE1"
```

and to close the file when finished by

```
DISK CLOSE,6
```

We can use these last two statements to store a string received from the modem. The input from the modem would be

```
INPUT #1,A$
```

where the string A$ must have as its last character <RETURN> .

Combining these three statements into a program to write a single message on disk

```
1Ø  DISK OPEN,6, "FILE1"   :REM OPENS DISK (W/ONE BUFFER)
2Ø  INPUT #1,A$            :REM LISTENS TO MODEM
3Ø  PRINT #6,A$            :REM ECHOS TO DISK
4Ø  DISK CLOSE,6           :REM CLOSES DISK FILE
5Ø  END
```

Likewise, we could later recover the data by the program

```
1Ø  DISK OPEN,6, "FILE1"
2Ø  INPUT #6,A$
3Ø  PRINT #2,A$
```

```
40  DISK CLOSE,6
50  END
```

In this problem we have written and read sequentially. If we modify the program to accept multiple messages, they would be stored sequentially, one after another.

You may inspect the sequential disk file by

```
RUN"SEQLST"
```

which provides a listing of the file when you give the information requested. The computer responds

```
SEQUENTIAL FILE LISTER
TYPE A  CONTROL C  TO STOP
FILE NAME?
```

You respond with the file name of a sequential file

```
FILE1
```

and a listing of the file will be printed. Upon reaching the end of the disk file, the message

```
ERR #D ERROR IN 100
```

will indicate completion of the listing.

Caution: if you use the SEQLST utility to inspect files which have BASIC programs stored in them, the display will look different than the original text. The reason for this is that the BASIC program stores BASIC source programs in a shorthand, called a tokenized form.

Another popular way is to transfer the disk file (let's say it was stored on track 39) by the CALL statement

```
DISK!"CALL D300=39,1"
```

which writes the file contents onto the middle of the CRT screen.

Note that some apparent garbage will be additionally printed here due to the unused portion of the disk file being printed, too.

If you wish to handle data in a random order, for example extracting the 20th data item from a file, it is not necessary to read the 19 prior data items. The use of random data items, also called records, is particularly useful when you examine a large set of data. Such data might be a set of customer accounts, a checking account history, or even temperature records for given days. In all these cases, the need arises to extract a specific record, without looking at all the prior records.

To aid in understanding the handling of random records, visualize a pointer which marks the start of a record. The GET command moves this pointer at the start of a given record. For example,

```
DISK GET,Ø
```

places the pointer in front of the first record. Similarly,

```
DISK GET,5
```

places the pointer in front of the sixth record. This method makes it easy to locate a record on the disk, however, it is wasteful of disk storage capability.

Each record uses a large disk area (128 bytes). The value of 128 bytes is preset by the operating system.

We may terminate a random (not sequential) input record by the PUT command. This will close the present record from further input.

A simple program to write three records on disk file "SCRTCH" and then GET the second record from that file, would be

```
 10  REM PROGRAM WRITE TEST
 20  REM OPEN THE DISK FILE SCRTCH
 30  DISK OPEN, 6, "SCRTCH"
 40  REM LOOP THREE TIMES TO END OF LOOP
 50  FOR TIME=1 TO 3
 60  REM PLACE 128 BYTE RECORDS ON DISK BY
 70  REM  (A) POSITION POINTER WITH A GET COMMAND
 80  REM  (B) PASSING THE MESSAGE TO THE DISK BY PRINT COMMAND
 90  REM  (C) CAUSE THE RECORD TO BE WRITTEN BY PUT COMMAND
100  DISK GET, TIME-1
110  INPUT #2,A$ : REM TYPE IN ANY PHRASE FROM KEYBOARD
120  PRINT #6,A$ : REM PLACE IN MESSAGE BUFFER
130  DISK PUT    : REM TRANSFER MESSAGE BUFFER TO DISK
140  NEXT TIME
150  REM END OF LOOP
160  RCRD=2
170  DISK GET,RCRD-1 : REM POINTER AT START OF RECORD 2
180  INPUT #6,A$     : REM READ DISK'S SECOND RECORD
190  PRINT #2,A$     : REM AND OUTPUT TO CRT (TERMINAL)
200  DISK CLOSE,6
210  END
```

The use of sequential and random disk files permits simpler control and bookkeeping than the CALL and SAVE or LOAD and PUT commands which we used for earlier file handling. This is one difference between record handling as compared to file handling.

5. More Devices

Memory can also be treated as a device. When we accept data from memory (Random Access Memory or RAM) as the input device, the DOS uses the address found in locations 238A (low address half) hexadecimal and 238B (high address half) hexadecimal) to determine what memory region to use. After each input, the address is incremented by one location. Memory, as an output device, is specified by the contents of 2391 (low address half) hexadecimal and 2392 (high address half) hexadecimal.

To load the address of memory to be used as an input device into 238A and 238B, and also load the address memory to be used as an output device into 2391 and 2392, DOS provides the command

MEM mmmm,nnnn

mmmm is the address of memory to be regarded as an input device (its starting address) and nnnn is the address of memory to be regarded as an output device (its starting address). For example,

MEM 5ØØØ,55ØØ

would load the locations

	Location Dec	Location Hex	Contents
Input Address	9Ø98	2384	ØØ
	9Ø99	238B	5Ø
Output Address	91Ø5	2391	ØØ
	91Ø6	2392	55

which establishes memory locations 5ØØØ and up to be used as an input device and locations 55ØØ and up to be used as an output device. No end of these memory regions is specified, so the user is cautioned in their use.

There are 256 selectable characters which are available for your use. The 256 characters, selected from a larger possible set, provide versatile graphics without heavy demands for memory.

The memory selected for storing the screen image is from 53248 to 55295 decimal. The color selected for each symbol is stored in another set of memory locations from 57344 to 59391. The locations for storing color values are 4096 locations beyond the location for the corresponding symbol. (Since 16 colors are available, only 4 bit (half byte) storage is provided.) You might regard memory as an image of the screen.

A work sheet is provided in the appendix to make an easier task of screen picture layout.

Display of any image is achieved by placing (in BASIC, using the "POKE" command) the character value and its color in the desired locations. For example, the BASIC program to place a blue "X" in the middle region of a 64 by 32 character display, at location 54302 (D41E hexadecimal) would be

```
10  POKE  56832,5 : REM TURN COLOR ON, SOUND OFF
20  POKE  54302,188 : REM MID SCREEN LOCATION  54303
30  REM  SYMBOL 181 IS AN X
40  POKE  58348,8 : REM COLOR NUMBER 8 IS BLUE
```

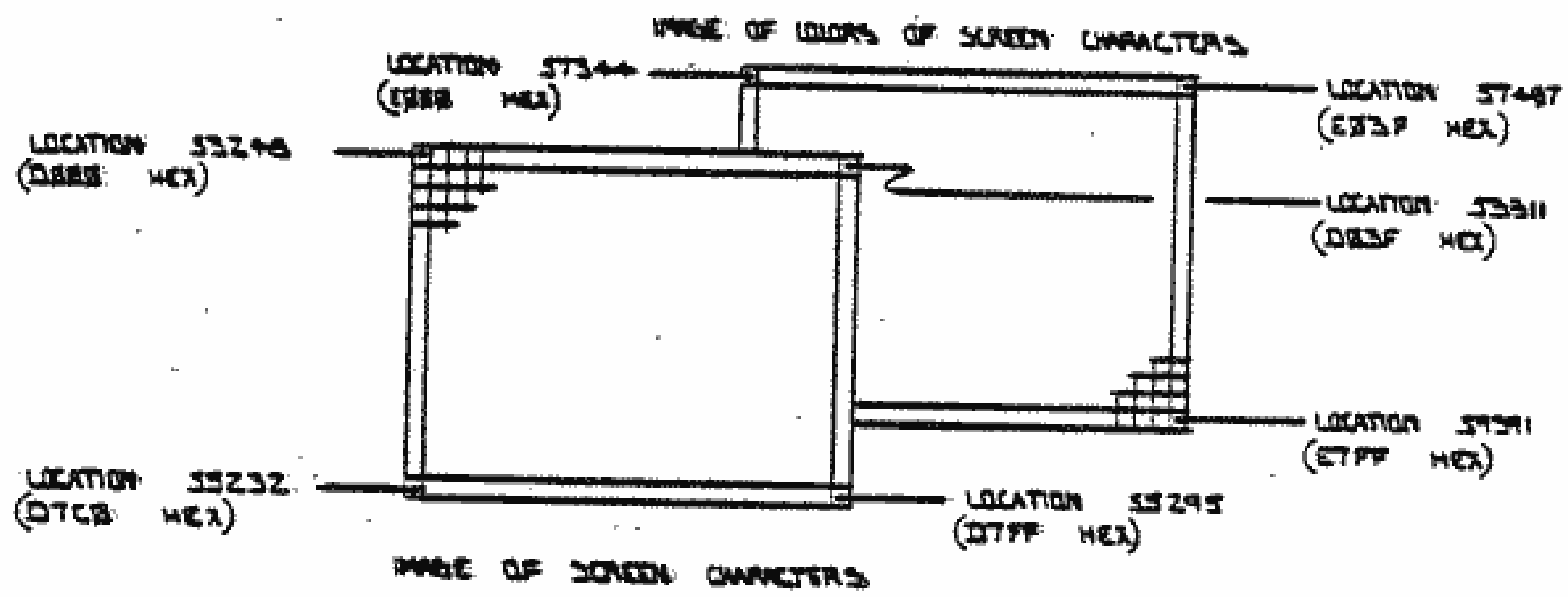
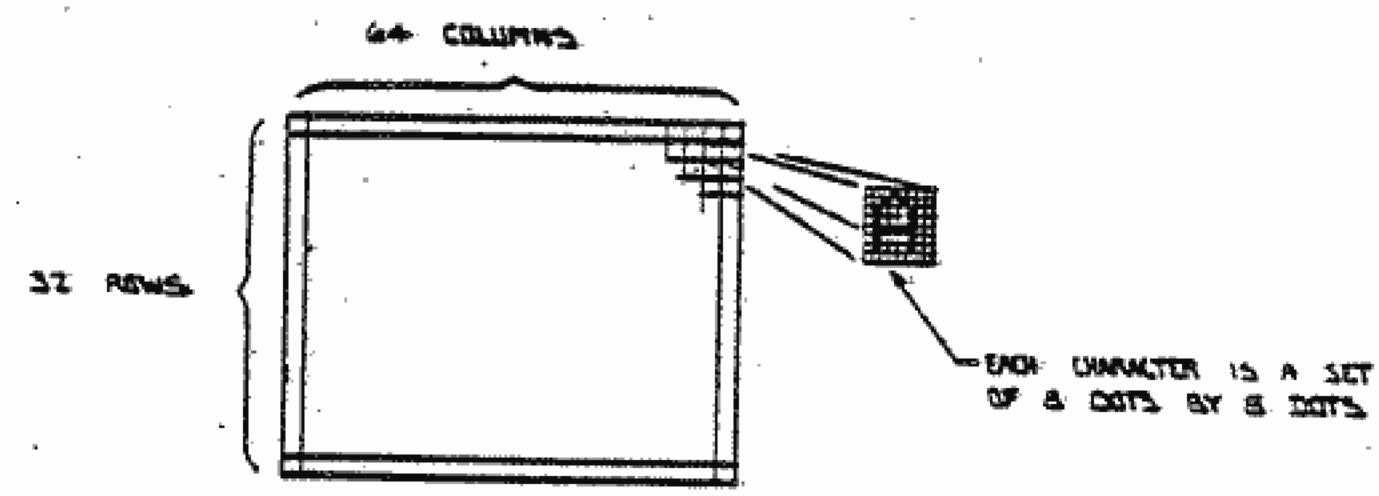


IMAGE OF COLORS OF SCREEN CHARACTERS
LOCATION 57344 (E000 HEX)
LOCATION 57407 (E03F HEX)
LOCATION 53248 (D000 HEX)
LOCATION 53311 (D03F HEX)
LOCATION 59391 (E7FF HEX)
LOCATION 55232 (D7C0 HEX)
LOCATION 55295 (D7FF HEX)
IMAGE OF SCREEN CHARACTERS

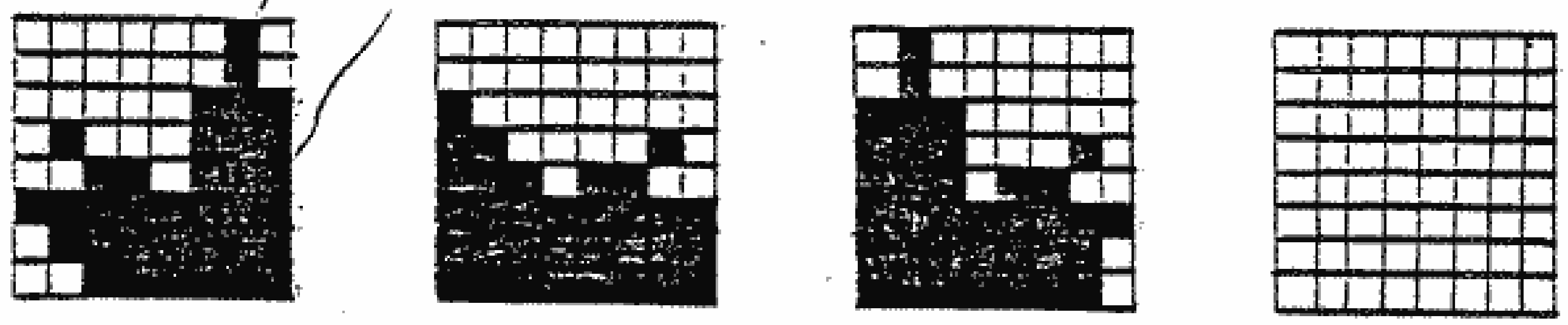

64 COLUMNS
32 ROWS
EACH CHARACTER IS A SET OF 8 DOTS BY 8 DOTS

Our color selections must be made from the list

Decimal Value	Color Selected
0	Yellow
1	Inverted Yellow
2	Red
3	Inverted Red
4	Green
5	Inverted Green
6	Olive Green
7	Inverted Olive Green
8	Blue
9	Inverted Blue
10	Purple
11	Inverted Purple
12	Sky Blue
13	Inverted Sky Blue
14	Black
15	Inverted Black (no color)

An inverted color is a black background with the symbol in color. Each of the 32 by 64 cells can be colored. To improve viewing, only the center two-thirds of the screen is used for graphics. For any line, the left and right border's color is the same as the last cell on the line (rightmost). The right border wraps its color around to the left border. The cell immediately before the leftmost (addressable) cell has the same color as the leftmost cell.

To illustrate the color choices, we shall try a program that places the symbol numbers 181, 182, 179, 180 (the shape of a ship in that order) into adjacent locations.



181 182 180 96

We shall display this ship across four columns for 16 times. Each time we shall change the color. Our program would be

```
 10  POKE 56832,5 : REM SET UP COLOR ON, SOUND OFF
 20  ST=53248     : REM START AT UPPER LEFT
 30  C=ST+4096    : REM COLOR AT 4096 BEYOND SCREEN LOCATION
 40  FOR RW 0 TO 32 : REM ROW INCREMENT LOOP
 50  FOR CM 0 TO 63 STEP 4 : REM COLUMN INCREMENT LOOP
 60  D=RW*64+CM   : REM COMPUTE SCREEN DISPLACEMENT
 70  POKE ST+D+0,181 : REM SHIP USES 4 CELLS
 80  POKE ST+D+1,182
 90  POKE ST+D+2,180
100  POKE ST+D+3,96
110  FOR I+1 TO 3
120  POKE CM+D+I,INT(CM/4) : REM SAME COLOR FOR WHOLE SHIP
130  NEXT I
140  NEXT CM
150  NEXT RW
160  GOTO 20
```

Since we have looped the program on itself, we use <CONTROL C> to exit this program.

Examining the possible character fonts in the appendix shows a wide variety of useful images for your own program sources.

We have examined a simple method to read the key closures without disturbing the video display. This method can be extended to the keypad and joystick accessories, which are merely extensions of the keyboard.

By using similar programs, interactive games and their displays are easily controlled. The complexity of the most involved game does not require any more than the example we just examined.

Some special purpose keys should be mentioned.

1) SL - the SHIFT LOCK key forces upper case letters to be printed on the CRT. It should be depressed prior to bringing up your system or running BASIC. Unlike a typewriter, however, the numbers will be printed normally. If you wish to type the symbols above the numbers, press the <SHIFT> key simultaneously with the desired character. The SHIFT LOCK key is used for normal entry. It should be released only for use of lower case letters, and then reset.

2) BREAK - resets the computer any time after the system is powered up.

3) SPACE BAR - provides a space when pressed.

4) RETURN - must be pressed after a line is typed. The previously typed line is then entered into computer memory.

5) CONTROL C - press <CONTROL> while simultaneously pressing C. Program listing or executing is interrupted, and the message

   BREAK IN LINE XXX

   is printed.

6) SHIFT O - press <SHIFT> first while simultaneously pressing O. The last character typed is erased. By the way, O is the letter "oh"; Ø will represent the number "zero". You do not type the slash. It is just to make reading easier.

7) SHIFT P - press <SHIFT> first while simultaneously pressing P. The current line being typed will be erased. The symbol '@" will be displayed. The effect will be to erase the line typed and enter a <RETURN> and <LINE FEED>.

8) D - When pressed after <BREAK> , causes initialization of the computer and boots the operating system from disk.

9) M - When pressed after <BREAK>, causes initialization of the computer. The computer is then in its machine language monitor.

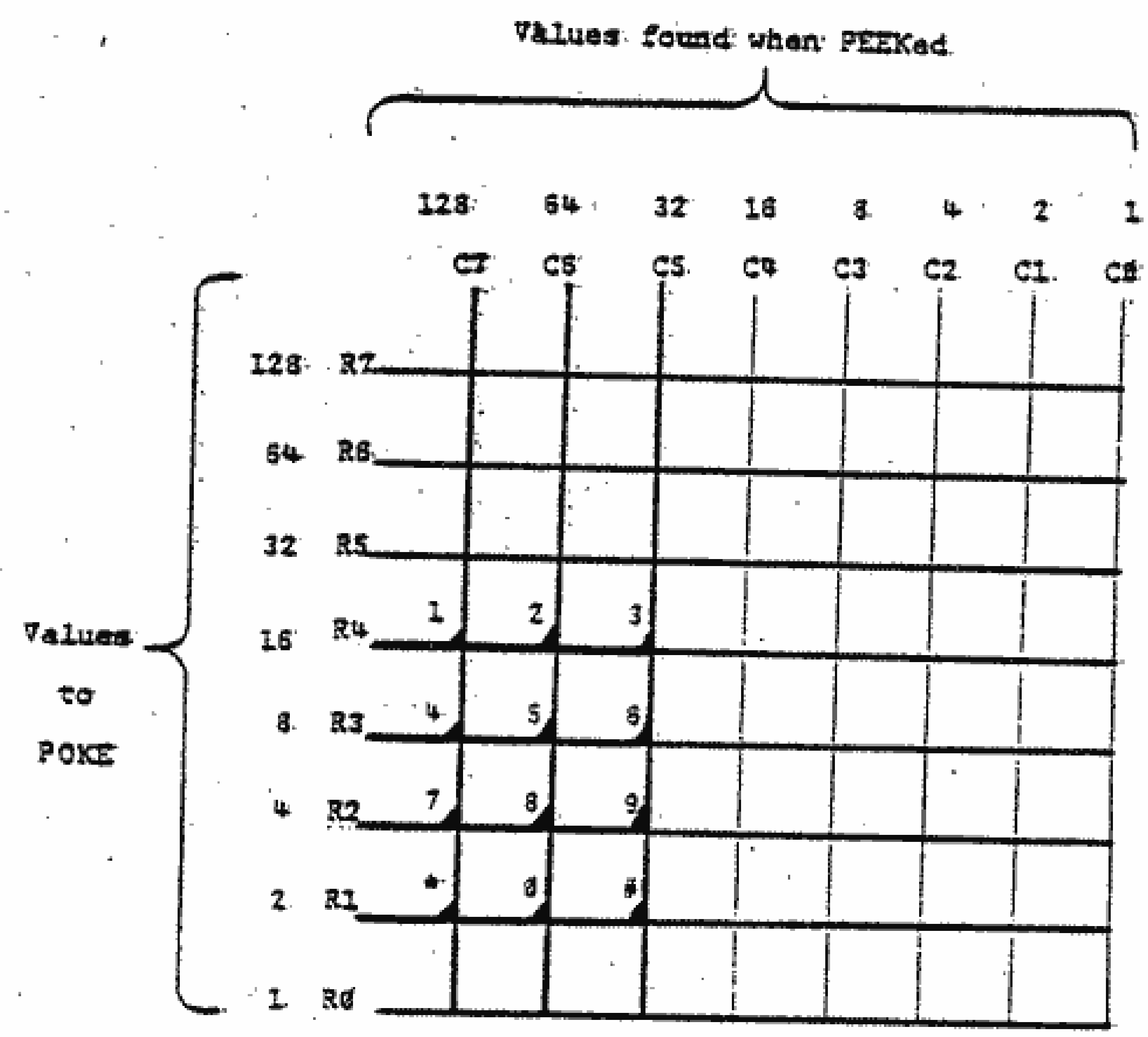
With this agreed notation, let's write a program!

9. Keypad

The keypads are merely extensions of the keyboard as are the joysticks. They can be read in the same manner as the keyboard is read by the computer.

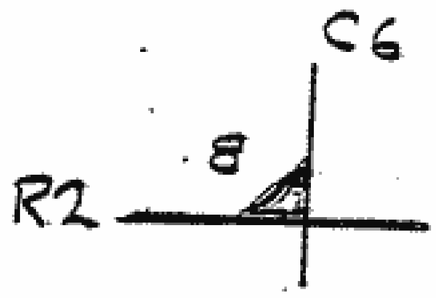
Prior to reading the keypad, we must disable <CONTROL C>, with a POKE 2Ø73,96.

Let's examine how keypad A is connected. Keypad A consists of a set of wires which correspond to keyboard rows shown labeled as R1 to R4. These are shown superimposed on the keyboard rows RØ to R7. In the same manner, the keypad A contains wires corresponding to keyboard columns C5 to C7 out of the total keyboard set of columns CØ to C7. When a key is pressed, a connection is made between the row and column where the switch is.



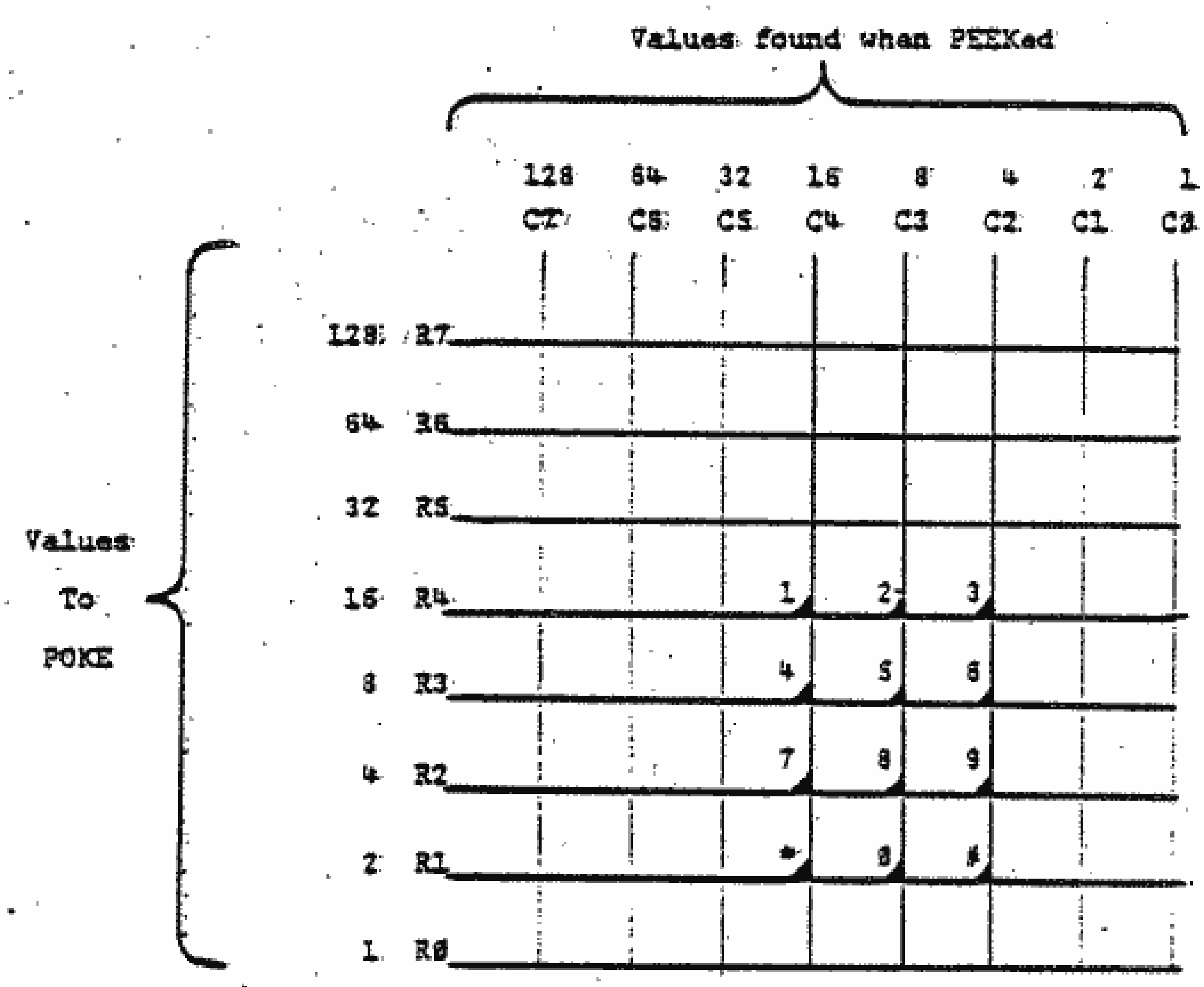
Keypad A

A cross-over point for keypad A will be indicated as (Row 2 and Column 6 joined when press key for symbol "8")



with the key symbol next to the shaded region.

Likewise, keypad B is connected as



Keypad B

Since keypad A is connected across R4, R3, R2 and R1, we can ignore the other rows by examining these lines only. The values of R4, R3, R2, and R1 are 16, 8, 4, and 2, respectively.

We can detect the symbol 8 (located at the intersection of Row 2 and Column 6 on keypad A) by setting Row 2 via

```
10 POKE 57Ø88,4
```

where 4 is the value POKEd to activate Row 2. We can sense Column 6 (value associated with column 6 is 64) by

```
2Ø TEST = PEEK(57Ø88)
3Ø IF TEST = 64 THEN GOTO 1ØØØ
```

where statement 1ØØØ takes care of the case when the 8 value is found.

A short program to read the key "8" or the key "#" and print the respective key is:

```
1Ø  REM KEYPAD TEST
2Ø  REM DISABLE <CONTROL C>
3Ø  CTRLC=2Ø73: DISABL=96: POKE CTRLC,DISABL
4Ø  REM NOW SET POINTER TO KEYPAD LOCATION
5Ø  P=57Ø88: R2=4: C6=64: R1=2: C5=32
1ØØ A$=" "
11Ø POKE P,R2 : REM TEST FOR 8
12Ø IF PEEK (P)=C6 THEN A$="8" : REM ON R2,C6
13Ø POKE P,R1 : REM TEST FOR "#"
14Ø IF PEEK (P)=C5 THEN A$="#" : REM ON R1,C5
```

10. Joystick

The joysticks provide realistic and convenient input devices for games and control. They are connected to the system as shown in Figure 1. The joysticks provide a digital signal when they are connected and enabled.

Prior to using the joysticks (or keypads) the <CONTROL C> command must be disabled by

```
POKE 2Ø73,96
```

The enabling of joystick A is done by

```
POKE 57Ø88,128 : REM - ENABLE JOYSTICK A
```

and joystick B is enabled by

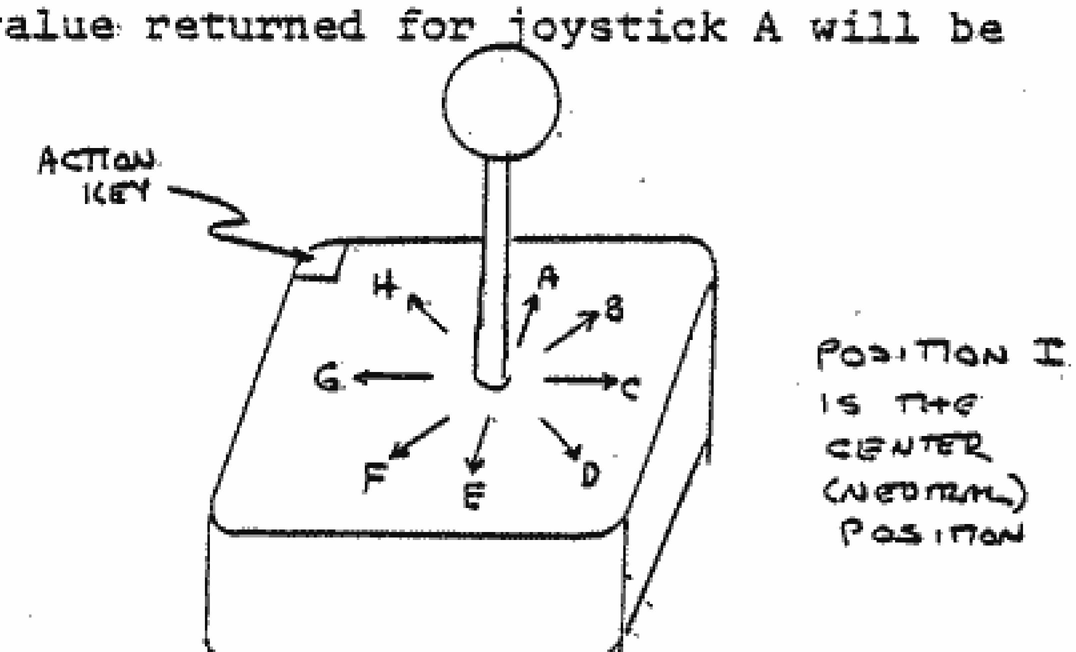
```
POKE 57Ø88,16 : REM - ENABLE JOYSTICK B
```

Only one joystick can be enabled at a time.

The joystick position can be read using the PEEK command. The value found using the PEEK command must be ANDed with a constant, depending on which joystick is used, to obtain a value for the specific joystick position. The constants used are 31 for joystick A and 248 for joystick B. For example

```
APOSIT=PEEK(57Ø88) AND 31
```

will return a value for APOSIT (A's position) which indicates the joystick position. If the "ACTION" KEY is not depressed, the value returned for joystick A will be



Joystick Position	Joystick A: Action Key Depressed Decimal Value Returned	Joystick A: Action Key Not Depressed Decimal Value Returned	Joystick B: Action Key Depressed Decimal Value Returned	Joystick B: Action Key Not Depressed Decimal Value Returned
A	16	17	32	160
B	20	21	48	176
C	4	5	16	144
D	12	13	80	208
E	8	9	64	192
F	10	11	72	200
G	2	3	8	136
H	18	19	40	168
I	0	1	0	128

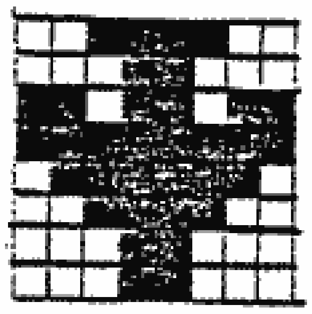
With the action key depressed, 1 has been added to the "action key not depressed" value for joystick A.

When joystick B is enabled, the corresponding values returned are returned to

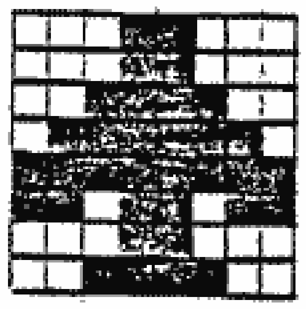
```
BPOSIT=PEEK(57088) AND 248
```

The "action key depressed" causes 128 to be added to the "action key not depressed" value for joystick B.

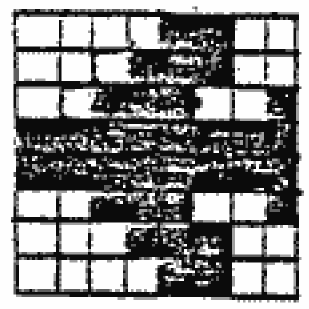
Let's try a sample program. We'll use the airplane figures



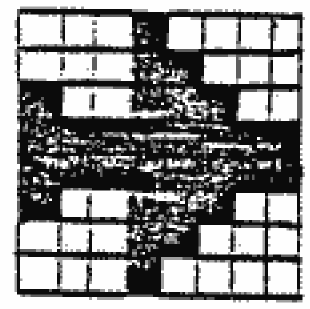
238



236



239



237

to move about the screen. Let's place the plane in the screen center to start at location 534Ø4 (D42Ø hexadecimal). We'll ignore clearing the screen, for example, simply leaving it in B & W with 64 characters per line and the sound off, by typing

```
1Ø  POKE 56832,1
```

We'll put the original plane on the mid-screen by

```
2Ø  POKE 543Ø4,236
```

Since we are B & W, no color is given. We shall use the "ACTION" button to quit (exit) the program. We shall use the logic shown in Figure 7.

Flow Chart for Airplane and Joystick

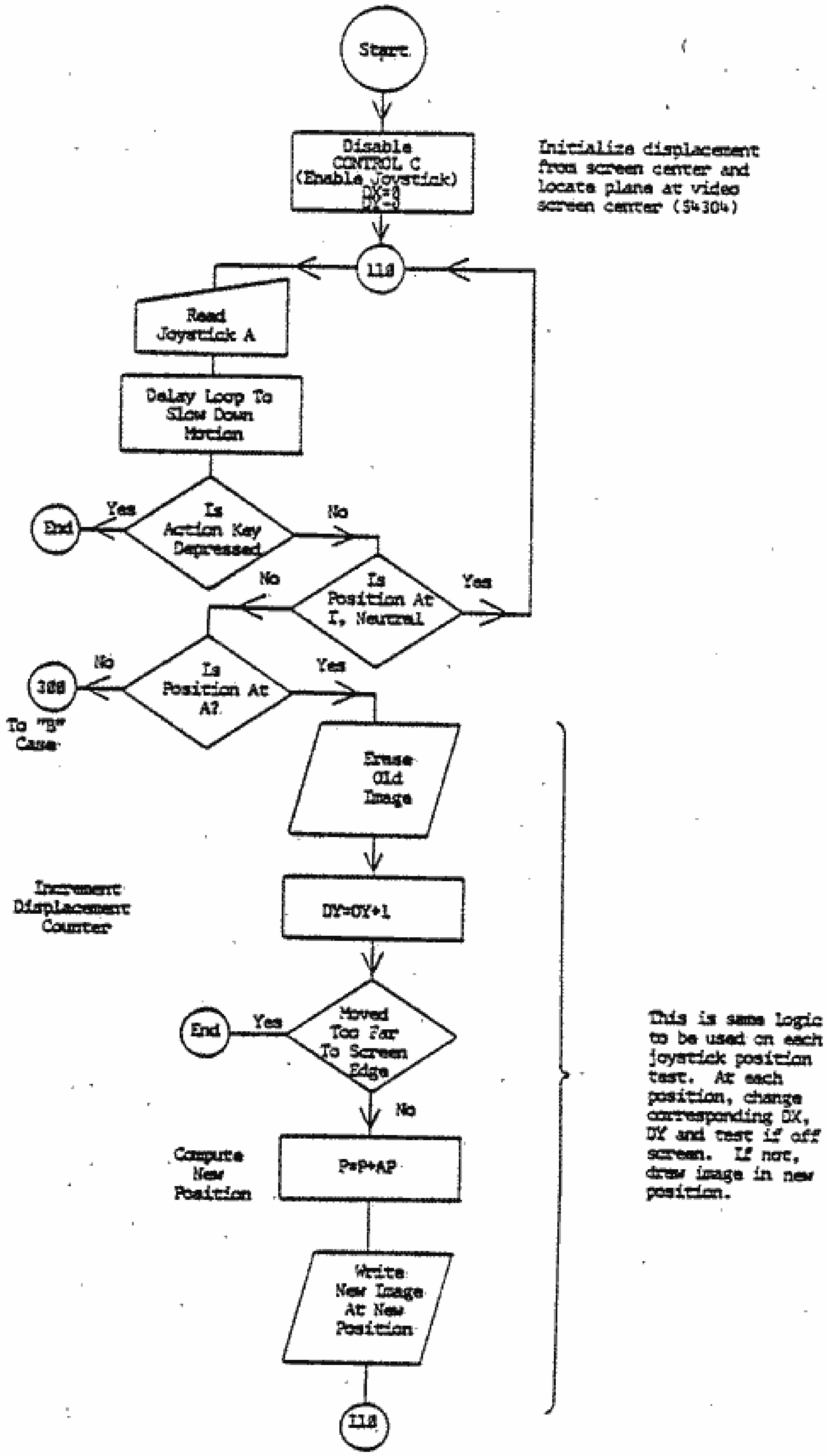


Figure 7.

The program to implement this flowgraph is

```
1Ø    POKE 2Ø73,96 : REM DISABLE  <CONTROL C>
2Ø    AP=-64:BP=-62:CP=+1:DP=66  :REM SCREEN POSITION DISPLACEMENTS
3Ø    EP=64:FP=62:GP=-1:HP=-66:IP=Ø  :REM RESULTING FROM JOYSTICK
                                      POSITION
35    REM
4Ø    A=16:B=2Ø:C=4:D=12           :REM CODE VALUES FOR
5Ø    E=8:F=1Ø:G=2:H=18:I=Ø        :REM JOYSTICK POSITION
55    REM
6Ø    POKE 57Ø88,128               :REM ENABLE JOYSTICK A
7Ø    BLANK=96                     :REM SCREEN SYMBOL FOR BLANK
8Ø    DX=Ø:DY=Ø
9Ø    P=54304                      :REM MIDSCREEN START
1ØØ   POKE P,236
11Ø   R=PEEK(57Ø88) AND 31
12Ø   FOR K=1 TO 2ØØ:NEXT K        :REM DELAY LOOP
13Ø   IF(R/2-INT(R/2)) >.1 THEN GOTO 9ØØØ :REM QUIT IF ACTION KEY
135   REM                          :REM DEPRESSED (ODD VALUE R)
14Ø   IF R=IP THEN GOTO 11Ø
15Ø   IF R =A THEN GOTO 17Ø
16Ø   GOTO 3ØØ
17Ø   POKE P, BLANK                :REM - ERASE OLD IMAGE
18Ø   DY=DY+1
19Ø   IF ABS(DY) >16 THEN GOTO 9ØØØ :REM IF OFF SCREEN, QUIT
2ØØ   P=P+AP
21Ø   POKE P,236                   :REM "A" POSITION IS UPWARD PLANE
22Ø   GOTO 11Ø
3ØØ   IF R=B THEN GOTO 32Ø         :REM"B" CASE
```

```
310  GOTO 400
320  POKE P,BLANK
330  DY=DY+1:DX=DX+1
340  IF ABS(DX) >30 OR ABS(DY) >16 THEN GOTO 9000
350  P=P+BP
360  POKE P,237
370  GOTO 110
400  IF R=C THEN GOTO 420         :REM "C" CASE
410  GOTO 500
420  POKE P,BLANK
430  DX=DX+1
440  IF ABS)DX) >30 THEN GOTO 9000
450  P=P+CP
460  POKE P,237
470  GOTO 110
500  IF R=D THEN GOTO 520         :REM "D" CASE
510  GOTO 600
520  POKE P,BLANK
530  DX=DX+1:DY=DY-1
540  IF ABS(DX) > 30 OR ABS(DY) >16 THEN GOTO 9000
550  P=P+DP
560  POKE P,238
570  GOTO 110
600  IF R=E THEN GOTO 620         :REM "E" CASE
610  GOTO 700
620  POKE P,BLANK
630  DY=DY-1
640  IF ABS(DY) >16 THEN GOTO 9000
```

```
650  P=P+EP
660  POKE P,238
670  GOTO 110
700  IF R+F THEN GOTO 720         :REM "F" CASE
710  GOTO 800
720  POKE P,BLANK
730  DX=DX-1:DY=DY-1
740  IF ABS(DX) >30 OR ABS(DY) >16 THEN GOTO 9000
750  P=P+FP
760  POKE P,239
770  GOTO 110
800  IF R=G THEN GOTO 820         :REM "G" CASE
810  GOTO 900
820  POKE P,BLANK
830  DX=DX-1
840  IF ABS(DX) >30 THEN GOTO 9000
850  P=P+GP
860  POKE P,239
870  GOTO 110
900  IF R=H THEN GOTO 920         :REM "H" CASE
910  GOTO 110
920  POKE P,BLANK
930  DX=DX-1: DY=DY+1
940  IF ABS(DX) >30 OR ABS(DY) >16 THEN GOTO 9000
950  P=P+HP
960  POKE P,239
970  GOTO 110
9000 END
```

Though the example appears to be long, it is repeated use of the same tests and operations, in blocks of less than 10 instructions. We now have a nucleus of programs to implement our own games!

11. Real Time Control of Devices

The heart of AC control lies in being able to run programs of immediate interest while a secondary program sits "in the background" waiting to be run. At periodic intervals, set by a hardware timer, the primary program ("in the foreground" of the computer's attention) is exited, at which time the secondary or background task is serviced. Then the primary task is re-entered and execution picked up where it was previously left. Note that all of this is happening very rapidly.

Background tasks are simple, rapidly computed programs which require periodic attention. Updating a clock display or checking home security status are examples of such a task.

The operating system OS-65D V3.2 HC contains a program "RTMON" which decides which program, foreground or background, should be run.

In addition, there are three programs, AC, AC1 and AC2 which support the use of AC control accessories. The program AC contains no buffers; AC1 contains 1 buffer; AC2 contains 2 buffers. If you make copies of this disk, you should copy only the version of this AC control program (AC, AC1 or AC2) which you need.

Your demonstration disk will show you some examples of the usefulness of AC control.

To write your own programs, the following sections will show the features

1) time of day clock
2) timing events
3) AC control and home security switches

Later sections will show how to integrate these features into a real-time system for your personal applications.

12. Time of Day Clock

The clock is a basic building block of a real time control system. The time of day clock does not have to be enabled; it runs continually under the 3.1 HC operating system. To set the time of day clock, we set hours in location 9480, minutes in 9481, and seconds in 9482. The commands are

```
POKE 9480,H    (H=number of hours)
POKE 9481,M    (M=number of minutes)
POKE 9482,S    (S=number of seconds)
```

The clock is a 24 hour clock which resets the time at 23.59:59 back to 0:0:0. Location 9483 holds the count of the number of 24 hour periods (i.e. days) which have been counted.

Time is read by the PEEK command. For example:

```
10  REM INPUT TIME TO SET CLOCK
20  INPUT "HOURS, MINUTES, SECONDS";H,M,S
30  POKE 9480,H:POKE 9481,M;POKE 9482,S
40  REM NOW TO PRINT OUT TIME
50  H=PEEK(9480):M=PEEK(9481):S=PEEK(9482)
60  PRINT H;"":";M":";S;"LOCAL TIME"
70  END
```

will permit setting the time, then displaying the time. Replacing statement 70 with

```
70  GOTO 50
```

will continually print the time.

13. Count Down Timer

The count down timer is an event timer which functions like an egg timer. A time count is loaded (set into) the timer which then counts down to zero.

Rather than have to check the current value of the timer count, a flag is raised when the count reaches zero.

To operate the time of day clock, the count down timer is loaded with the hours in location 224, the minutes in location 225, and the seconds in location 226.

Starting the count down timer is accomplished by placing a 1 in location 223. Disabling the count down timer (turning it off) requires a Ø in location 223.

A program to set the count down timer and start it running is

```
10  POKE 223,0
20  INPUT "HOURS, MIN, SEC COUNTDOWN";H,M,S
30  POKE 224,H
40  POKE 225,M
50  POKE 226,S
60  REM NOW START TIMER
70  POKE 223,1
```

A program could check the one location, 223, to determine if the hours, minutes, and seconds had elapsed by

```
80  TEST=PEEK(223)
90  IF TEST=0 THEN GOTO 1000
100 GOTO 90
```

The real value of the timer, however, lies in its ability to request the services of the real time monitor, RTMON. RTMON permits interrupting user programs when the count down timer reaches zero. This switching of priorities from one program to an interrupting program allows flexible programming. These uses will be discussed after we have looked at some other devices and features available for home and appliance control.

14. External Switches, Alarms, Or Indicators

In AC control and home security systems, we often need to sense switch openings or closings. Relay contacts might indicate an air-conditioner "on" for an energy management system; an open window might be read as a set of open contacts to a home security system. Your imagination is the limit.

The C4P system provides (in the AC-12 package) the ability to sense 48 separate remote contact-pairs. Each of these contact-pairs (lines) is to be at either Ø volts or 5 volts (standard TTL levels). When these lines are computer driven (used for output) a maximum of two TTL devices can be driven at a time. If devices other than OSI peripheral devices are used, you are cautioned to use good circuit practices in interfacing circuits.

The input lines are grouped as 6 sets of 8 lines (8X6=48), or 6 input registers. Associated with each input register (group of 8 lines) are a mask register (tells which of the 8 lines to ignore) and an active state register (tells whether a 5 volt or Ø volt signal is to be the chosen active state). The state of each line can be sensed by examining the register bit which reflects the state of the connected line. In the case of windows, for example, we might wish to identify the active state as an open window in one program but in a different program we want the active state to reflect a closed window. Which one we want will depend on our program.

The associated registers, i.e., the mask register and active state register, are used by the real time monitor, RTMON, to systematically scan the input lines. When an input line becomes active, RTMON's services are requested (in the same manner as the count down timer requested service). Once again, we will put off discussion of how RTMON uses these associated registers until we have first examined the hardware which is used to support RTMON.

The associated registers are memory locations which are examined to determine how we interpret switch positions. In contrast, the hardware registers directly indicate line status, 5 volts or Ø volts. The hardware registers also indicate whether a set of lines is to receive signals (be read) or whether output signals should be sent to turn on/off devices (to be written to).

External switches which can be used to provide 5 volt or Ø volt are connected (through back panel connectors, Figure 1) to a Peripheral Interface Adapter (PIA). The PIA presents groups of input lines for input or output of signals. These input or output lines are addressed in groups of 8 lines. The PIA is a single integrated circuit. Its organization and use are best explained in terms of its addressing, i.e., where the computer looks to input or output data. For this purpose, we create a map.

15. PIA Data Register

A map of the hardware registers used for input and output is

Hex Location	Decimal Location	Data Register (Bit 7 … Ø)	Control Register	Decimal Location	Hex Location
C7Ø4	5Ø948	Port 1A			
			CTRL Register For Port 1A	5Ø949	C7Ø5
C7Ø6	5Ø95Ø	Port 1B			
			CTRL Register For Port 1B	5Ø951	C7Ø7
C7Ø8	5Ø952	Port 2A			
			CTRL Register For Port 2A	5Ø953	C7Ø9
C7ØA	5Ø954	Port 2B			
			CTRL Register For Port 2B	5Ø955	C7ØB
C7ØC	5Ø956	Port 3A			
			CTRL Register For Port 3A	5Ø957	C7ØD
C7ØE	5Ø958	Port 3B			
			CTRL Register For Port 3B	5Ø959	C7ØF

Each port A, port B pair is called a Peripheral Interface Adapter or PIA. These ports provide a way to enter data from the outside world into the computer and to respond with computer generated signals to the outside. The PIA also holds or latches these input and output signals until the computer is ready to receive them (for input) or until the outside devices can utilize them (for output). Each of the two ports on a PIA (port A and port B) contain 8 lines which may be individually used for input or output.

The CA-12 option contains three PIA's. The AC-12 is connected to the C4P computer by a 16 pin connector, J2, shown in Figure 1. External devices are connected to the three sets of input port pairs. Since three sets of port A-port B pairs are accommodated (each port 8 bits wide), we have 3*2X8=48 lines available for external connection.

The operating system will initialize the scan of PIA's to include a complete CA-12 option group of PIA's as a default. Scanning fewer PIA's or scanning the PIA at 63232 decimal (F700 hex) will require making the changes (POKEs) which we have just illustrated.

For example, to scan all 48 lines starting at 50948 decimal (C704 hex) all six data registers (ports 1A, 1B, 2A, 2B, 3A, 3B) must be scanned along with six control registers. Therefore, we must load location 8902 decimal with 12-1=11 (the number of scanned registers minus one). These POKEs can be accomplished as

```
POKE  8902,11 : REM LOOK AT ALL 6 DATA AND 6 CONTROL REGISTERS
POKE  8909,4  : REM LOWER HALF OF C704 PIA PORT ADDRESS
POKE  8910,199: REM SINCE C7 hex=199 decimal
```

(Only decimal values may be used with POKEs.)
With these POKEs, RTMON will check for an active state.

We have looked at the connections to the PIA. Let us now look at the operation of the PIA. The ports (port A and port B) serve two purposes. Each port accommodates input or output signals. Additionally, these port A and port B pairs serve as data direction registers. When serving as a data direction register, the port specifies which bits serve as input and which serve as output bits. The action of the port, whether it serves as an input/output port or as a data direction register, is set by yet another register, called the control register. A control register is associated with each port. If the control register is POKEd with zeros, then the port serves as a data direction register.

When the control register is POKEd with a 4, the port reverts to its data handling function. By using a data port to serve as a data direction register, the number of hardware connections is reduced. We pay a price of increased complexity in understanding its function. To illustrate, for example, to use the PIA to read port 1A at location 5Ø948 (C7Ø4 hex), the steps are

1) POKE 5Ø949,Ø
   This address, one beyond the PIA port 1A address, is the control register for port 1A. A zero in the control register will allow the use of the PIA port 1A address for its alternate use, designating which bits are input or output (called a data direction register). A 1 indicates output, a zero an input. At the completion of this POKE, the control register contains

   5Ø949 | ØØØØ ØØØØ |

   and the port 1A will serve as a data direction register. Therefore, the command

2) POKE 5Ø948,127
   will place the bit pattern Ø111 1111 into the data direction register. The data direction register will now be

   5Ø948 | Ø111 1111 |

   Bit 7, the leftmost bit of the data direction register contains a Ø indicating that its corresponding line will be an input line. The other register bits (bits Ø to 6) are 1's, indicating that their corresponding data lines will serve as output lines.

3) We are ready to revert the PIA port 1A to its data handling function. This is achieved by

   POKE 5Ø949,4

   which commands the control register for port 1A to perform its I/O function.

4) Bit 7, the leftmost bit, was previously set as an output bit in step 2. We can set this output to a high value by

   POKE 5Ø948,64

   This is a bit pattern 1ØØØ ØØØØ. The data register (the alternate function of the port) will now contain

   5Ø948 | 1ØØØ ØØØØ |

Likewise, we could have set bit 7 to a zero by

POKE 5Ø948,Ø

5) If we wished to read bit 6, which was designated as an input bit, we could have

BIT6 = PEEK (5Ø948) AND 64

where 64 has a bit pattern Ø1ØØ ØØØØ. The 1 in the bit pattern corresponds to the desired line. To the user, location 5Ø948 appears as

	7	6	5	4	3	2	1	Ø	bit
5Ø948	X	1 or Ø	X	X	X	X	X	X	

where X indicates that we don't care about the value. By ANDing the contents of 5Ø948 with the value

Ø 1 Ø Ø Ø Ø Ø Ø

only the value of bit 6 will be examined. If bit 6 of 5Ø948 is a zero, then BIT6=Ø; if bit 6 is 1, then BIT6=64. Testing for zero or non-zero value of BIT6 provides a convenient programming test to determine the bit 6 input line state.

The socket pin connections are shown in the appendix; socket mating information is also provided.

A short program to make all lines for port 1A read (input) lines and all lines for port 1B into write (output) lines follows:

```
5   REM PIA INITIALIZATION SUBROUTINE AT 1ØØØ
1Ø  GOSUB 1ØØØ
2Ø  INPUT "SIDE (A OR B)",C$
3Ø  IF C$="A"GOTO 1ØØ
4Ø  IF C$="B"GOTO 2ØØ
5Ø  GOTO 2Ø
1ØØ IF A$="I"GOTO 15Ø
11Ø INPUT "OUTPUT TO A";K
12Ø POKE X,K
13Ø GOTO 2Ø
15Ø PRINT"INPUT TO A IS";PEEK (X)
```

```
 160  GOTO 20
 200  IF B$="I"GOTO 250
 210  INPUT "OUTPUT TO B";K
 220  POKE X+2,K
 230  GOTO 20
 250  PRINT "INPUT TO B IS";PEEK (X+2)
 260  GOTO 20
1000  INPUT "STARTING ADDRESS OF PIA";X
1010  INPUT "A SIDE I OR "OK";A$
1020  INPUT "B SIDE I OR "OK";B$
1030  POKE X+1,0:POKE X+3,0 : REM SETTING CTRL REGISTER TO ZERO
1040  IF A$="I" THEN POKE X,0: PERMITS SETTING DATA DIRECTION
                               REGISTER
1042  IF A$="I" THEN GOTO 1050
1045  POKE X,255 : REM IF NOT INPUT, THEN SET AS OUTPUT
1050  IF B$="I" THEN POKE X+2,0
1052  IF B$="I" THEN GOTO 1060
1055  POKE X+2, 255
1060  POKE X+1,4:POKE X+3,4 : REM CTRL REGISTER TO FORCE I/O
1070  RETURN
```

Multiple lines may be checked at one time.

The home security system addressed at 63232 (F700 hex) is also a PIA port. It is one of two ports. Two ports (of 8 bits each) are available, with the first 4 bits being reserved as:

Car Alarm → Bit 2
Intruder Alarm → Bit 1
Misc. Alarm → Bit 3
Fire Alarm → Bit Ø

Location	(Hex)	Bit 7 6 5 4 3 2 1 Ø
63232	F7ØØ	Port A
63233	F7Ø1	CTRL A
63234	F7Ø2	Port B
63235	F7Ø3	CTRL B

A program to handle this device is similar to the previous programs. For example, to check for a fire alarm

```
1Ø   REM SET PORT A AS INPUT, LOOK AT BIT Ø, THE FIRE ALARM BIT
2Ø   POKE 63233,Ø : POKE 63232,1 : POKE 63233,4
3Ø   IF PEEK (63232) = Ø THEN GOTO 1ØØ
40   GOTO 2Ø
```

This program segment will continually look at the input port and check for the bit assigned by OSI to fire alarm checks.

16. Real Time Monitor, RTMON

The Real Time Monitor, RTMON, acts as a watchdog, responding when either the count down timer counts down to zero or a PIA device is sensed to be "active". The internal computer hardware interrupts processing every 400 milliseconds (.4 seconds) to update the count down timer and the time of day clock.

Should either the count down timer go to zero or a PIA device line go "active", then computer control is immediately passed to the program, RTMON. Within the program RTMON, you may decide what action is to be taken.

A typical RTMON program should deactivate the timer by

```
POKE 223,0
```

This allows servicing the interrupt without having the timer time out. This would avoid two interrupts occuring simultaneously; however, this uncertainty of occurrence accounts for only a few microseconds. Examining the timer contents and the PIA lines of interest will determine whether a PIA or the timer requested service. Before exiting RTMON the program should

```
POKE 222,1
```

to re-enable RTMON so that RTMON can be recalled by future interrupts. If we do not have any further programs to return to from RTMON, then we can terminate RTMON with a return to BASIC by

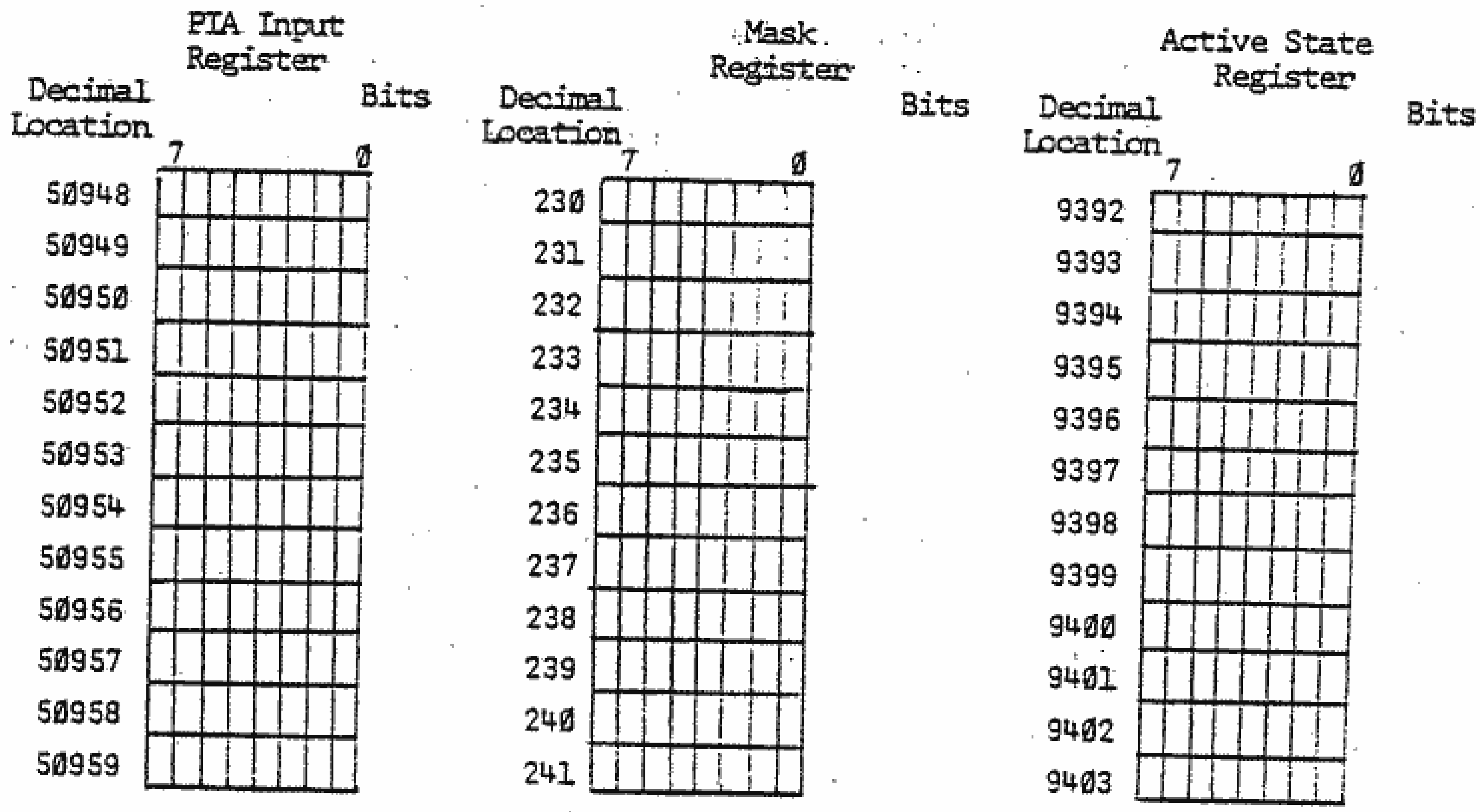
```
RUN"BEXEC*" : END
```

The operating system will then turn control over to the BASIC interpreter.

Within the operating system (specifically the OS-65D V3.1 HC, Home Control Operating System), certain provisions are made for monitoring and responding to all PIA lines. These special provisions are made for the devices hung on the 48 lines from 5Ø948 to 5Ø958 (C7Ø4 to C7ØE hex) and for the 16 lines at 63232 and 63234 (F7ØØ and F7Ø2 hex).

To sense an "active" state on a PIA line, each register of the PIA is matched to two associated registers. A "mask register" (this indicates which bits of the PIA are to be monitored) and an "active state register" (this indicates whether a high level, '1', is the active state or a low level) 'Ø', is the active state. RTMON will be called by the operating system is a bit is not masked out and has reached its alarm state.

Let's look at these memory locations as a map



A Ø bit implies ignore the corresponding line. A 1 bit implies watch the corresponding line.

A Ø bit means look for a Ø (low) as the active state in the corresponding PIA input register. Also, see example for additional restrictions.

We shall ignore a bit in the PIA data registers when the corresponding bit in the mask register is a Ø. If the mask register bit is set to 1, then the corresponding PIA data register bit is examined.

If we chose to ignore a bit from a PIA register (data or control register)(by placing a Ø in the corresponding bit position in the mask register) then, we must place a 1 in the corresponding position of the active state register.

We choose which registers to scan by POKEing (placing) the address of the first register to be scanned in 89Ø9 and 891Ø. The lower half of the address (low byte) is POKEd in 89Ø9 (22CD hex) and the upper half of the address (high byte) is POKEd in 891Ø decimal (22CE hex). Place the number of registers to be scanned (minus one) in location 89Ø2 decimal (22C6 hex).

For example, if we wish to examine bit 6 of the PIA port at location 5Ø948 decimal (C7Ø4 hex), we should place the bit pattern Ø1ØØ ØØØØ (64 decimal) into the mask register at 23Ø decimal (E6 hex). This will force ignoring all but bit 6. The corresponding active state register at 9392 decimal (24BØ hex) should contain the bit pattern 1Ø11 1111 (183 decimal, B7 hex) if we want a Ø to indicate the active state. If a 1 is to be the bit 6 active state, then the bit pattern should be 1111 1111 (255 decimal, FF hex).

If all 8 bits of a mask register are zero (ignore all data bits) then no special value must be placed in the active state register since it will be totally ignored.

Though you will probably not want or need to examine the control registers for each port, this ability is provided (you may want to examine the interrupt lines of the PIA, for example).

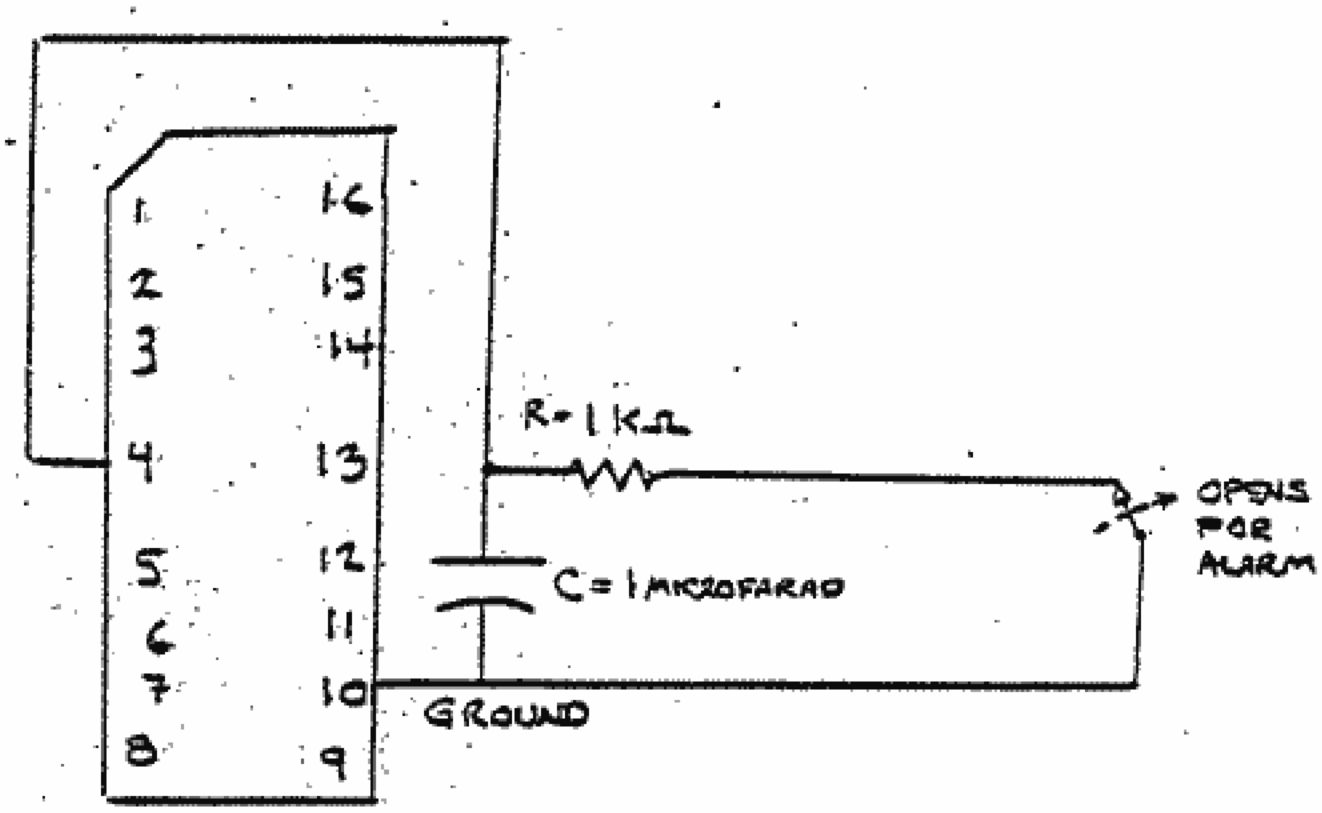
If you do not specify which set of PIA ports to scan, the operating system will choose 5Ø948 decimal (C7Ø4 hex) as the starting value. This is the choice of the CA-12 option PIA's.

17. A Greenhouse Example

Let's try an AC control example which monitors a home greenhouse. While we enjoy normal use of the computer, we wish to have a low termperature alarm available "in the background." If the temperature should drop below a preset value, we wish to be informed of the event. Additionally, we'd like to have an hourly signal sent to the greenhouse to spray the plants.

Both timer and alarm tasks are well suited to our C4P system. These tasks are performed by the real time monitor, RTMON.

A circuit which will accomplish the alarm function is



J3 (OF FIGURE 1)

The other available connector pinouts are shown in the appendix. The selected circuit grounds the PIA input PA3 at address 63232 decimal (F700 hex). When the temperature triggers the alarm, a bimetalic thermostat connection opens and the PIA goes to a high state (due to its internal power connections).

A 1 microfarad capacitor in the alarm circuit minimizes noise pickup, while the 1K ohm resistor minimizes noise currents picked up on the long wires leading to the greenhouse. Twisted pair shielded wire, though more costly than unshielded wire, is advised for extended applications.

No warranty or liability by use of this (or other) user circuit is to be inferred. Good practice is encouraged.

Let's break the software part of this problem into smaller pieces. First, we should set the hourly timer in the main program, to get started. Also, we need to set up the PIA addresses and masks which the real time monitor will scan.

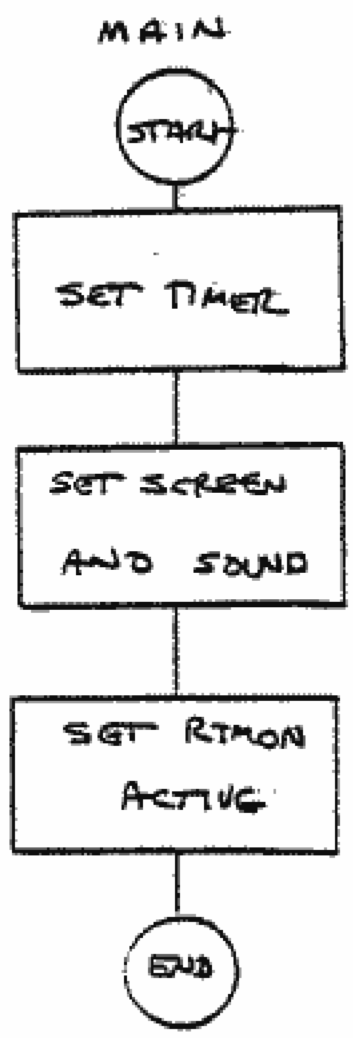
Once initialized, the 3.1 HC will scan the timer and the PIA line control to the alarm circuit. When the timer runs down to zero, the monitor will reset the timer. Also, if the temperature alarm has been tripped, the monitor will react. In either case, alarm or timer, the monitor, RTMON, will be reset before leaving the RTMON program.

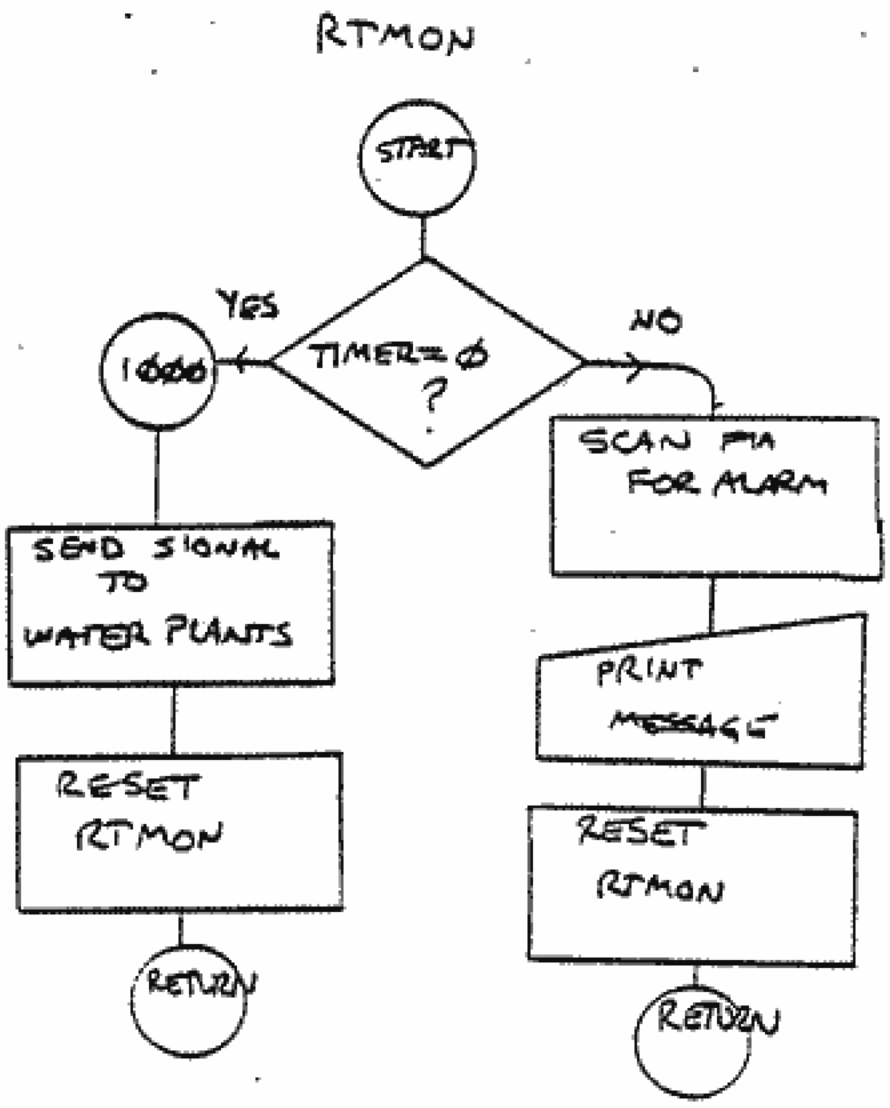
Because the program RTMON is resident on disk and is brought into the user's work space at the alarm or timer run out time, the current contents of the work space will be destroyed. If any data must be retained, they must be stored periodically on a file on disk. If these data are needed, this provision to save them should be made. Generally, this loss of data or running program is not considered to be a problem, as returning the work space to BASIC with the BEXEC* program would place the user in command of all the computer's resources. The previously running program could be called again with only slight inconvenience.

To use RTMON, it is necessary to have a main program and the real time monitor, RTMON. The main program (or possibly the program BEXEC*) will initialize and activate RTMON. The main program will be the normally operating program. Only when an event (timer times out or PIA line is alarmed) occurs will RTMON come into play. Otherwise, operations of RTMON is transparent to the user.

In this example, RTMON will interrupt the operation of the main program when our greenhouse needs help. The causes for a request for help are (a) the temperature exceeds a preset value on a thermostat or (b) the hour between waterings is up, and the sprinkler must be turned on.

In the blocks, we have the programs





```
10 REM RTMON PROGRAM FOR GREENHOUSE
20 IF PEEK(223)=0 THEN GOTO 1000
25 REM CHECK IF TIMER AT ONE HOUR ELAPASED?
30 IF PEEK(9392)<>247 THEN GOTO 200
35 REM 247 IS NON-ALARM STATE
200 REM SOUND TONE ALARM AND PRINT ALARM
210 PRINT"TEMPERATURE ALARM"
215 PRINT PEEK(9392)
220 POKE 57089,INT(49152/440)
230 REM TONE IS IN HEARING RANGE
240 FOR T=1 TO 500:NEXT T: REM DELAY LOOP
250 POKE 57089,1  :REM TURN OFF ALARM
260 POKE 222,1:REM ENABLE RTMON
270 PRINT"IT WAS TEMPERATURE":GOTO 1090
1000 REM NEED TO ACTIVATE SPRAYER
1010 REM TO WATER PLANTS.  USE A
1020 REM SINGLE PULSE FOR THIS DEVICE.
1025 POKE 223,0:REM MAKE SURE TIMER OFF
1030 POKE 224,1:REM RESET HOURS
1040 POKE 225,0:REM RESET MIN
1050 POKE 226,0:REM RESET SECONDS
1055 PRINT "TIMER TEST"
1060 POKE 223,1     :REM SET TIMER
1070 POKE 222,1     :REM ENABLE RTMON
1080 PRINT"AT END WE ENABLE RTMON"
1090 END
```

```
10 REM MAIN PROGRAM TO SET UP GREENHOUSE
20 REM
30 POKE 223,0:REM DISABLE TIMER
40 POKE 224,1:REM SET HOURS TO 1
50 POKE 225,0:REM MINUTES AT 0
60 POKE 225,0:REM SECONDS AT 0
65 REM WATER EVERY HOUR
70 POKE 223,1:REM ACTIVATE TIMER
80 POKE 56832,7:REM TURN ON SOUND AND COLOR
81 REM SETUP PIA
82 POKE  63233,0
83 POKE 63232,0:REM LOOK FOR INPUTS
84 POKE 63233,4:REM REVERT TO DATA HANDLING
90 POKE 8909,0:REM ADDRESS OF PIA
100 POKE 8910,247:REM ADDRESS OF PIA
110 POKE 8902,0:REM LOOK AT FIRST REG. PORT A ONLY
120 POKE 230,8:REM MASKS 0000 1000 FOR LOOK AT BIT 3
130 POKE 9392,247:REM MASKS 1111 0111 FOR BIT 4
135 REM 247 DECIMAL IS F7 HEX.  SELECT F700 PIA.
140 REM ACTIVE LOW
150 POKE 222,1:REM ENABLE RTMON
160 PRINT    "ENABLE RTMON IN MAIN"
170 END
```

For this example, we shall generate a short 44Ø hertz tone pulse to alert the user. The remark, statement 1Ø2Ø, might be replaced with ACTL commands to turn on and off a watering fixture or an output to a PIA to create a pulse. Which you choose would depend on the watering device characteristics.

The overall flow chart is adequate to follow the detailed program listing.

If the user wished a more detailed response to the alarms, minor modifications within the program framework would achieve these actions.

If the user wishes to try these programs, files to store "MAIN" and "RTMON" should be created. Then, these programs could be retained for future use on disk.

RTMON would be stored (after being typed in) by

DISK!"PU RTMON"

and the main program (after typing in) by

DISK!"PU MAIN"

The program would be initiated after receiving control of the computer from BEXEC* by entering

RUN"MAIN"

18. BEXEC*

BEXEC* is the program which links the operating system and the end user programs. It is run by the operating system prior to turning control of the computer over to the user. BEXEC* typically provides setting critical parameters, such as specifying the input and output devices, and disabling or enabling certain entries, such as the <CONTROL C> entry to permit interrupting user programs. The demonstration disks and the operating system disks each have a program called BEXEC*. You may use these versions, if you wish, by copying the BEXEC* program for use in your own user program development. However, you will often wish to set some initial parameter (i.e., POKE some location) or run some initial program (such as a screen clearing program) prior to reverting to input to the BASIC system.

Let's start with an example of one:

```
10 REM BASIC EXECUTIVE
20 REM
24 REM SETUP INFLAG & OUFLAG FROM DEFAUL
25 X=PEEK(10950): POKE 8993,X: POKE 8994,X
30 PRINT : PRINT "BASIC EXECUTIVE FOR OS-65D VERSION 3.0" : PRINT
40 PRINT "13 OCT 1978 RELEASE"
50 GOTO 100
60 PRINT : INPUT "FUNCTION";A$
70 IF A$="CHANGE" THEN RUN "CHANGE"
80 IF A$="DIR"    THEN RUN "DIR   "
90 IF A$="UNLOCK" THEN 10000
100 PRINT
110 PRINT "FUNCTIONS AVAILABLE:"
120 PRINT "   CHANGE - ALTER WORKSPACE LIMITS"
130 PRINT "   DIR    - PRINT DIRECTORY"
140 PRINT "   UNLOCK - UNLOCK SYSTEM FROM END USER MODIFICATIONS"
150 GOTO 60
10000 REM
10010 REM UNLOCK SYSTEM
10020 REM
10030 REM REPLACE "NEW" AND "LIST"
10040 POKE 741,76 : POKE 750,78
10050 REM
10060 REM ENABLE CONTROL-C
10070 POKE 2073,173
10080 REM
10090 REM DISABLE "REDO FROM START"
10100 POKE 2893,55 : POKE 2894,8
10110 PRINT : PRINT "SYSTEM OPEN" : END
```

The BEXEC* program shown sets the input and output devices to be the keyboard and video display and prompts the user to use the DIRectory or CHANGE utilities. If these utilities are not requested, the editing and debugging features of "NEW", "LIST", and <CONTROL C> are enabled. In certain programs (such as the example used in the section on Joystick use), you may wish to disable these optional utilities prior to running your programs. BEXEC* provides the ideal time to take care of these housekeeping functions.

Demonstration or game disks often require special provisions to be made. BEXEC* provides the opportunity to make these changes, including the guiding of the user by program prompts. To simplify use of demo or game disk, it is often convenient to start the user in his/her program. For example to run a program (here called DEMO), the last statement in BEXEC* could be

```
RUN"DEMO"
```

In this manner, BEXEC* can take care of routine keyboard entries and simplify user response. As in most endeavors, simple is better.

19. Summary

Conclusion

In the preceding sections, we have looked at many devices and their use. Sensors (switches and alarms), keyboards and joysticks provide communication from the outside world. Tones, modems and printers provide communication from the computer to the user. Our C4P has the ability to respond intelligently to its environment.

By effective use of the disk, large programs can be broken into smaller programs and brought into memory as needed. Message and supervisory records can be kept on disk for future reference or processing.

These features, operating with the real time monitor, RTMON, provide rapid response through a wide range of devices, in a rapidly changing environment. By using the programming examples which we have shown in this manual, resilient operation can be expected, even in the event of unexpected data. To the user, the real time feature will provide the effect of two computers, one operating on tasks of immediate importance, the other monitoring the security and status of programs of background concern. The background program can assume a high priority when it is needed.

The ability to control, supported by OSI hardware and software, makes the personal computer a strong and able servant to your tasks.

Chapter VII

Advanced Techniques

By this time, you have written several BASIC programs and should be comfortable with your C4P system. This chapter will assume a familiarity with assembly and machine code programming. Borden's book How to Program Micorcomputers, available from your OSI distributor, and the two manuals, Ohio Scientific Extended Machine Language Monitor User's Manual and Ohio Scientific 65ØØ Assembler/Editor User's Manual, are convenient references. With these cautions, we shall try some assembly language or machine code programs.

Assembly language or machine code programs are more involved to write, since much of the detail is left to the programmer. In compensation, programs will run significantly faster and permit more versatility.

1. Machine Monitor, 65V

The machine monitor provides a simple way to examine and modify memory contents. Data or programs are entered using hexadecimal (base 16) notation. Programs must be entered in machine code using hexadecimal notation.

The machine monitor provides a simple command structure. You enter the machine monitor after typing <BREAK> when the C4P gives you the prompt

H/D/M?

You then type

M

The machine then responds with

ØØØØ XX

where XX are two hexadecimal characters. You are now in the machine monitor, displaying the contents of location ØØØØ.

To load a given location (address) with data or program, type a period

.

This will select the addressing mode. If you were already in the addressing mode, you will remain in the addressing mode. You may now type the desired address which you wish to enter. If an entry error is made, reentering the address will remove the old value.

To enter data into the selected memory location, you must transfer to the data entry mode. This is done by typing a slash

/

Data may now be entered as two hexadecimal characters. As in the address mode, an incorrect entry can be corrected by typing the correct value. To increment to the next sequential location, press

<RETURN>

When you have completed loading your program, you may execute the program at its starting address (for illustration, I'll use hexadecimal address Ø2ØØ) type the starting address and then the letter "G" as

.Ø2ØØG

(The period entry returned us to the address mode.) The program will start executing. (The machine monitor Goes to Ø2ØØ to start.)

Illustration

Let's load a program which places graphics characters 25Ø (hexadecimal FA) into mid video screen location 5432Ø (Hexadecimal D43Ø). An assembly language program and its machine code would be

Hex Location	Machine Code	Assembly Code	Comment
Ø2ØØ	A9	LDA #$FA	FA is symbol for eastward tank
Ø2Ø1	FA		
Ø2Ø2	8D		
Ø2Ø3	3Ø	STA $D43Ø	Tank to midscreen
Ø2Ø4	D4		
Ø2Ø5	EA	NOP	
Ø2Ø6	4C	JMP $Ø2Ø5	Jump back to NOP
Ø2Ø7	Ø5		
Ø2Ø8	Ø2		

This program should place an eastward point tank (character 25Ø) near mid video screen. The machine monitor instructions would be

<BREAK>

.Ø2ØØ

/A9 <RETURN>

FA <RETURN>

8D <RETURN>

3Ø <RETURN>

D4 <RETURN>

EA <RETURN>

4C <RETURN>

Ø5 <RETURN>

Ø2 <RETURN>

.Ø2ØØG

At this point, the tank should appear near mid video screen.

For the cassette user, the command L permits loading program from cassette. Upon typing L , all ASCII commands are accepted from the audio cassette rather than the keyboard. Cassettes are prepared with an auto-loading program at their beginning. Examples of this are the Extended Machine Code Monitor cassette and the Assembler/Editor cassette. When the program is loaded, the cassette playback unit may be rewound and turned off.

In summary, the Machine Monitor commands are

/ - Use Data Mode

. - Use Address Mode

G - Start execution at the address presently displayed on video screen.

L - Transfer control to the audio cassette.

Some of the hexadecimal locations which the Machine Monitor uses are

FEØØ - Start of Monitor (restart location)

FEØC - Restart with clear video screen, other Machine Monitor parameters unchanged

FE43 - Entry into Address Mode, with initialization bypassed

FE77 - Entry into Data Mode, with initialization bypassed

These entry points may be useful to incorporate within your other programs.

2. USR(X) Routine

We can combine the speed of machine code execution with the simplicity of BASIC using the USR(X) function. The linking of machine code and BASIC programs is accomplished by the single BASIC statement

```
X=USR(X)
```

The USR(X) function permits leaving the BASIC program, executing a machine language routine, and then returning to the original BASIC program. To call the USR(X) routine in BASIC, a pointer to the location of your USR(X) routine must have been stored. In our BASIC, these pointers are at 22FC hexadecimal (8956 decimal) for the low half of the hexadecimal address and 22FB hexadecimal (8955 decimal) for the high half of the hexadecimal address.

Typically, we shall want to protect the machine language (code) program by placing it in high memory. If we move BASIC's "end of memory" pointer to a value at least two pages (512 decimal words) down from the physical value of "end of memory", we can assure that this memory area is not used by any other routine. For example, on a 24K system (24576 decimal, 6000 hex) these limits would be

```
 24576
-  512
 24064
```

The equivalent calculation in hex is

```
 6000
-200
 5E00
```

Therefore, setting 5E00 hex as "end of memory" will give a 512 byte clear region for calculations. This "end of memory" value should be stored with the high order two hex digits in location 2300 hex (8960 decimal) i.e., POKE 8960,94.

Since we shall want to store the "end of memory" value with a POKE command in BASIC, let's convert 5E00 hex first

```
 5E   00   hexadecimal
 ↓    ↓
 94   00   decimal
```

Since the address of end of memory requires two bytes for storage, two POKEs are necessary. The POKE command requires decimal values as operands. Therefore, we must convert each half of the hex address into decimal, one half at a time. Conversion was accomplished by looking up the decimal conversion in the table provided in the appendix. The high order hex equivalent digits are stored by

```
POKE   8960 ,          94
end of memory pointer   high memory boundary
```

The lower half of the "end of memory" is assumed at the page end (ØØ).

Now choose the lower end of this now protected memory (above the official "end of memory") to store our USR(X) routine. Place the address of USR(X) in the location pointer to where BASIC expects the USR(X) address. The address of USR(X) can be loaded by using POKEs. We can POKE the two address parts of USR(X) into the location which stores USR(X)'s address by

```
POKE 8955,ØØ : REM - LOW BYTE OF USR(X) ADDRESS
POKE 8956,94 : REM - HI BYTE OF USR(X) ADDRESS
               REM INTO USR(X) POINTER
```

We now need to write a program, USR(X), to be stored in memory starting at 5EØØ hex (24Ø64 decimal). Please note that this last decimal value is the result of converting all four hex digits of 5EØØ at one time, rather than finding the decimal equivalent of each half of the address. The earlier conversions of half of the address were for storage convenience, and were not for evaluating the whole address value.

Example: A Screen Clearing Routine

To illustrate the USR(X) routine, we shall write a routine to clear the CRT terminal screen. We shall place the letter "A" at each screen position, sequentially to illustrate the speed of this routine. Of course, replacing the letter "A" with the symbol for a blank would produce a general screen clearing. This program is described by a flow chart in Figure 4 which is reduced to assembly language in Figure 9. In this example, our last statement is an RTS (return from subroutine), which returns us to the calling BASIC program.

In the example, we shall use the 65Ø2 microprocessor's accumulator as the register for data transfer. The X-register and the Y-register will be used as counter registers. This usage will be economical in terms of data transfer time, since the accumulator is the central point for transfer purposes. The X- and Y-registers are serviced with increment and decrement commands to aid counting operations.

By converting the hexadecimal machine code into decimal values, the code can be POKEd into the desired memory locations. This is a handy method to enter machine code routines while in BASIC. A BASIC program to store this machine code at the required locations is

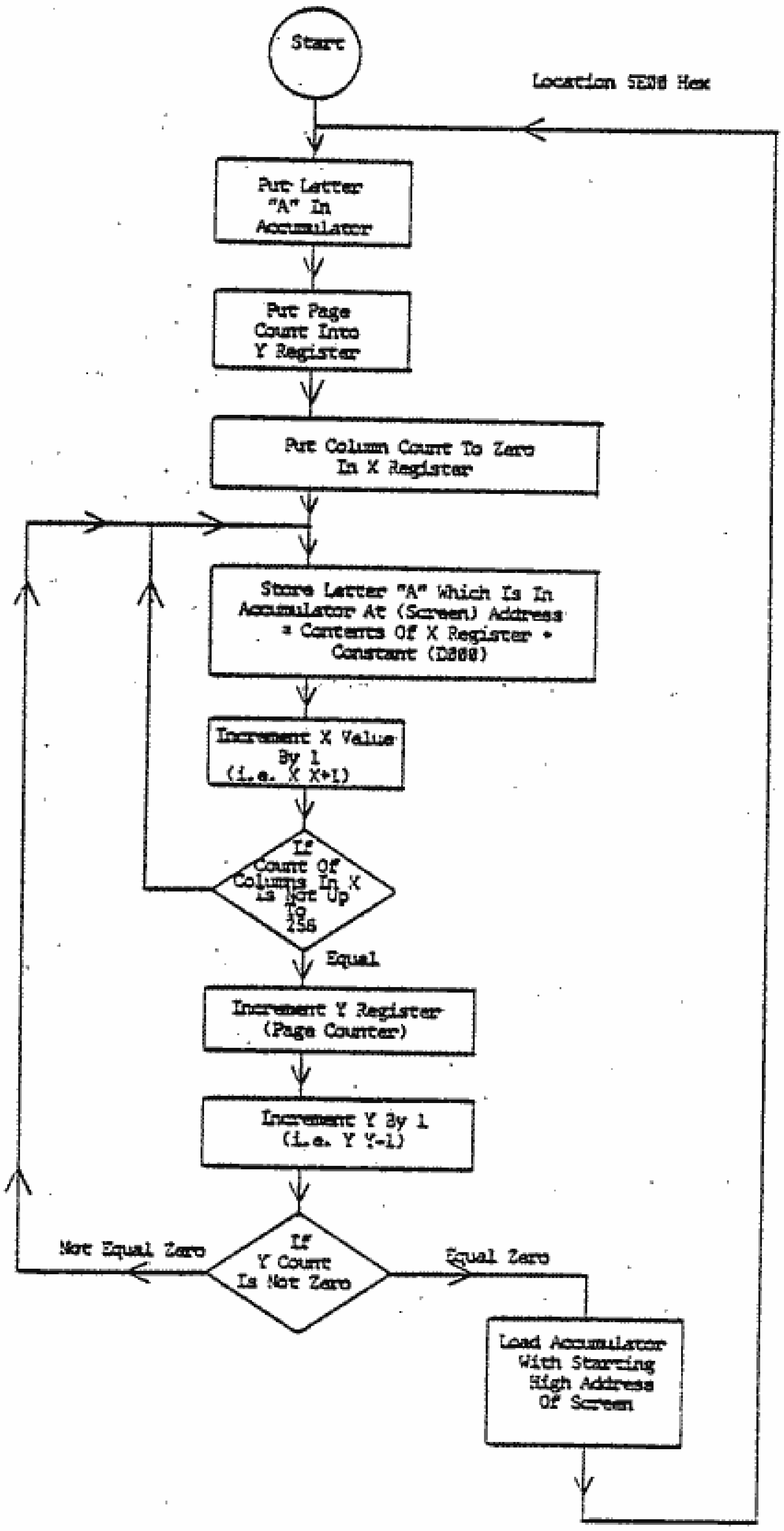

Start
Location 5E00 Hex
Put Letter "A" In Accumulator
Put Page Count Into Y Register
Put Column Count To Zero In X Register
Store Letter "A" Which Is In Accumulator At (Screen) Address = Contents Of X Register + Constant (D800)
Increment X Value By 1 (i.e. X X+1)
If Count Of Columns In X Is Not Up To 256
Equal
Increment Y Register (Page Counter)
Increment Y By 1 (i.e. Y Y-1)
If Y Count Is Not Zero
Not Equal Zero
Equal Zero
Load Accumulator With Starting High Address Of Screen

FIGURE 4

Hex Location	Decimal Location	Machine Code	Assembler Code	Comment
			*=$5EØØ	Set program counter to 5EØØ
5EØØ	24Ø64	A9 41 -	LDA #$41	Load accumulator with ASCII A
5EØ2	24Ø66	AØ Ø8 -		Load page count
5EØ4	24Ø68	A2 ØØ -	LDX #$ØØ	Load column counter at zero
5EØ6	24Ø7Ø	9D ØØ DØ	STA $DØØØ,X	Store "A" at each screen position
5EØ9	24Ø73	E8 - -	INX	Increment column on screen
5EØA	24Ø74	DØ FA -	BNG $5EØ6	If columns not complete, loop to store "A" again
5EØC	24Ø76	EE Ø8 5E	INC $5EØ8	If columns complete, increment page (4 line) counter
5EØF	24Ø79	88 - -	DEY	Decrement page count
5E1Ø	24Ø8Ø	DØ F4	BNG $5EØ6	If not complete page count, loop to store "A" again
5E12	24Ø82	A9 DØ -	LDA #$DØ	If pages complete, then reset screen address
5E14	24Ø84	8D Ø8 5E	STA $5EØ8	Restore operand of page count
5E17	24Ø87	6Ø - -	RTS	Go back to calling program

FIGURE 5

[redacted]

The machine code of Figure 9 (for sequential locations)

Hex Location	Decimal Location	Machine Code (Hexadecimal)	Machine Code (Decimal)
5EØØ	24Ø64	A9	169
5EØ1	24Ø65	41	65
5EØ2	24Ø66	AØ	16Ø
5EØ3	24Ø67	Ø8	8
5EØ4	24Ø68	A2	162
5EØ5	24Ø69	ØØ	Ø
5EØ6	24Ø7Ø	9D	157
5EØ7	24Ø71	ØØ	Ø
5EØ8	24Ø72	DØ	2Ø8
5EØ9	24Ø73	E8	232
5EØA	24Ø74	DØ	2Ø8
5EØB	24Ø75	FA	25Ø
5EØC	24Ø76	EE	238
5EØD	24Ø77	Ø8	8
5EØE	24Ø78	5E	94
5EØF	24Ø79	88	136
5E1Ø	24Ø8Ø	DØ	2Ø8
5E11	24Ø81	F4	244
5E12	24Ø82	A9	169
5E13	24Ø83	DØ	2Ø8
5E14	24Ø84	8D	141
5E15	24Ø85	Ø8	8
5E16	24Ø86	5E	94
5E17	24Ø87	6Ø	96

```
 5 REM CLEAR SCREEN PROGRAM
 1Ø RESTORE : REM SETS START OF DATA LIST
 2Ø P=24Ø64 : REM START AT 5EØØ HEX
 3Ø FOR I=1 TO 24
 4Ø READ M : POKE P,M
 5Ø P=P+1
 6Ø NEXT I
 7Ø DATA 169,64,16Ø,8,162
 8Ø DATA Ø,157,Ø,2Ø8,232
 9Ø DATA 2Ø8,25Ø,238,8,94
1ØØ DATA 136,2Ø8,244,169,2Ø8
11Ø DATA 141,8,94,96
12Ø END
RUN <RETURN>
```

Running this program places the desired machine code routine in memory. Now exit from BASIC by typing

```
EXIT <RETURN>
```

At this time, we can SAVE the machine code routine in high memory on disk. For example, if we use track 39 of our disk, starting at sector 1, by responding to the prompt

```
A*
SAVE 39,1=5EØØ/1 <RETURN>
```

This saves the program located at 5EØØ hexadecimal, starting on track 39 at sector 1 for 1 page (256 bytes). This program can be reloaded from disk by responding to the prompt as

```
A* CALL 5EØØ=39,1
```

The machine code routine would thus be read off track 39, sector 1 into RAM at 5EØØ. Running this screen clearing routine may be run as follows, reloading the program under BASIC. We may do this reloading under BASIC as

```
DISK!"CALL 5E00=39,1"
```

Therefore the BASIC program segment

```
    :
  90 POKE 8955,0 : POKE 8956,94 : REM SET USR(X) ENTRY POINT
 100 DISK!"CALL 5E00=39,1" : REM USR(X) STORED EARLY IN PROGRAM
    :
1000 X=USR(X) : REM SCREEN CLEARING ROUTINE INVOKED
```

This program segment, including USR(X), would provide a screen clear at far faster rate than possible with a BASIC program.

3. Using The Assembler

The preceding USR(X) program was shown in Assembly language. The C4P system supports an assembler. The Assembler/Editor could have been used for creating the program module which was SAVEd on disk.

To use the Assembler/Editor, boot up your system. Once in BASIC, request (after the OK prompt)

EXIT <RETURN>

Type (after the operating system prompts, shown underlined)

A* ASM <RETURN>

to get the Assembler, and enter your program (the same USR(X) program as before) after the Assembler prompt.

```
. 10  *=$5E00
. 20  LDA #$41
. 30  LDY #$08
. 40  LDX #$00
```

```
.50   STA $D000,X
.60   INX
.70   BNE $5E06
.80   INC $5E08
.90   DEY
.100  BNE $5E06
.110  LDA #$D0
.120  STA $5E08
.130  RTS
.A
```

The Assembler file will assemble your program and store it at 5E00 hexadecimal (24064 decimal). We have again obtained our machine code program in memory at 5E00 hexadecimal.

At this point, the use of the operating system to SAVE the program on disk would be the same as shown in the previous section, i.e., typing

SAVE 39,1=5E00/1 <RETURN>

would place our machine code on disk. Running the previous BASIC program segment

```
  90  POKE 8955,0 : POKE 8956,94
 100  DISK!"CALL 5E00=39,1"
1000  X=USR(X)
RUN
```

will result in the same screen clearing routine to be run.

The Assembly language listing provided the machine code needed for the USR(X) loading. Even if the Assembler is not used to create the USR(X) program module, the extensive editing routines of the Assembler/Editor encourage its use.

Note, for more detail on the Assembler/Editor see the Ohio Scientific Assembler/Editor Manual.

4. Executing a Disk Resident Machine Language Program

If you have a machine language program which you wish to use, there is an alternative to use of the BASIC routine

X=USR(X)

Assume we have a machine code program stored on a disk file named "FILE". The alternate method is used under the DOS. The response should be

A* <u>XQT FILE</u> <RETURN>

where FILE is the name of your machine language program on disk (or it can be the track number where it is stored).

Under BASIC, this is accomplished by

DISK!"XQT FILE"

In order to use the XQT command, however, some computer housekeeping is required first.

The XQT command brings a machine code program from disk and stores it at location 1292 decimal (3279 hexadecimal). When the machine code is stored on disk, some housekeeping is done. The first four bytes on the file used will contain a "header" which is labeling information provided by the assembler. The next (fifth) byte will contain how many tracks are to be loaded to contain the program. Then, from the sixth byte to the end of the file, the machine code program is stored.

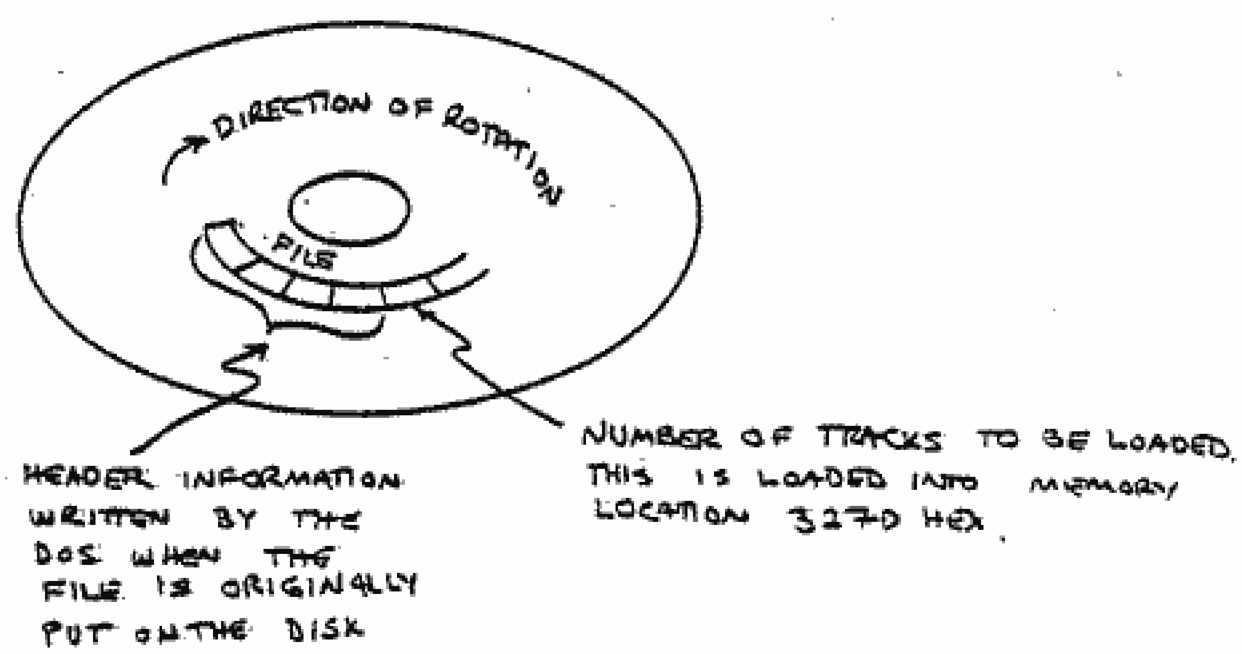
When a machine code file is loaded by the XQT command into memory starting at 1292 decimal (3279 hexadecimal), program control will have to skip over the header and track length information and start with the program stored at 12926 decimal (3273 hexadecimal).

Let's make a map of how the program is expected to appear on disk. Also, we'll make a map of how the file will be stored in memory.

XQT FILE STORAGE IN MEMORY

Decimal Location	Hex Location	Contents
12921	3279	
12922	327A	
12923	327B	File header created by Assembler
12924	327C	
12925	327D	Number of disk tracks to be loaded
12926	327E	Start of first program instruction
12927	327F	
:	:	

XQT FILE STORAGE ON DISK



With the housekeeping conventions established, let's start by creating a file called FILE1 which will contain an assembly language code. This program has not been converted into machine code yet. The program shown will store the message "ANY ASCII CHARACTERS" at locations D740 hexadecimal (55104 decimal) which is in the lower left hand of your video screen. We enter the program as follows

A* ASM <RETURN>

The computer will reply

OSI 6502 ASSEMBLER

COPYRIGHT 1976 BY OSI

Then we enter the assembly language code.

```
. 10         *=$327E                 { SET ORIGIN
. 20         LDX #0                  { SYMBOL COUNTER INITIALIZED
. 30 LBL1    LDA MSG,X
. 40         BEQ LBL2
. 50         STA $D740,X
. 60         INX
. 70         BNE LBL1
. 80 LBL2    JMP LBL2
. 90 MSG     .BYTE 'ANY ASCII CHARACTERS'
.100         .BYTE 0
.110         .END
```

We can store this in the previously created file - FILE1 - by typing

```
. !PUT FILE1
```

When this file is already on disk it could be recalled by typing

```
. !LOAD FILE1
```

In either case, we are not yet ready to assemble the source program, i.e., convert this program into machine code. When we do convert this program to machine code, we'll store the assembled (converted machine code) program at a location (address) 2ØØØ hexadecimal bytes displaced from the assembly language program. We must establish a memory displacement or offset, arbitrarily chosen here as 2ØØØ hexadecimal (valid for 24K machines) to be within memory available and above the region needed by the assembler program, by typing

. M2ØØØ

and then

. A3

The Assembler/Editor will now assemble the program and leave it at a location offset by 2ØØØ hexadecimal from the intended program origin. Now we can exit the assembler by typing

EXIT

We now wish to place the assembled (machine code) program at the final destination of 327E hexadecimal, which is where the XQT command command will place the first machine code program step. The Extended Monitor provides the means of relocating the program from location offset by 2ØØØ hex above the destination of 327E. The previously used region (327E hex and up) is no longer needed by the Assembler/Editor.

To invoke the extended monitor from the DOS type

EM

The extended monitor prompt is a colon. Type

: M 327E=527E,5298

The difference between the first two numbers is the offset value previously used. The last number, is one more than the last memory location required, all in hexadecimal. The Assembler/Editor provides the address of each instruction in the listing. By subtracting the last address from the first address in the listing, the hexadecimal length of the machine code (not including the last instruction) can be calculated. Shorter programs, of course, would require less memory.

We need to determine the integer number of tracks to store our machine code program. Each disk track can store 2K bytes of code (length of approximately 2ØØØ decimal).

Since our example is 19 hexadecimal in length (25 decimal), we require far less than one track (even if we add the five locations needed for the header). We put the information about the track requirement in location 327D by responding to the color prompt by

: @327D

The @ symbol is <SHIFT P> . The Extended Monitor permits you to store data in 327D following the prompt

327D/Ø1

We reply

Ø1

the number (two hexadecimal digit) of tracks required. The next response is

: EXIT <RETURN>

In earlier examples in the manual, we created files (called scratch files) for incidental use. Let's use one of those files named "SCRTCH" to store the machine code program. We store this machine code program by responding to the prompt

A* PUT FILE2

We can now verify the XQT command by responding to the prompt

A* XQT FILE2

Our message "ANY ASCII CHARACTERS" should appear on the screen.

The details of this section have been rather involved. By using machine code, we have had to accept the housekeeping responsibilities within the computer. In return, considerably faster running programs are obtained. Storage requirements of the programs are also reduced. If the speed and compactness of machine code is needed within the convenience of BASIC programming, the XQT command may prove worth the demands on the user.

5. Digital to Analog (D/A) Converter

For general applications, the C4P is equipped with a companding digital to analog converter (DAC). This DAC is coupled to the output through a capacitor. Therefore, only changing voltages can be observed. A constant voltage will be blocked by the capacitor. For example, a positively increasing signal from the DAC will appear at the output as a positive voltage. A decreasing signal from the DAC will appear as a negative voltage. The peak to peak voltage range is about 3 volts. (Brief maximum excursions of up to ±3 volts are possible at start up.)

Since the output of the DAC must change rapidly to pass through the capacitor coupling to the output, the program code which drives the DAC must be in machine code, rather than in BASIC.

A program to drive the DAC can be loaded under the machine monitor at boot up by responding to

H/D/M? M

Press the "period" (".") to enter the address mode and type

Ø3ØØ

as an address, then press the "slash" ("/") to alter the memory locations. Enter the two digit hex code at the addresses indicated

Address		
Ø3ØØ	E8 <RETURN>	Increment X
Ø3Ø1	8E <RETURN>	Store X at location $DFØ1
Ø3Ø2	Ø1 <RETURN>	
Ø3Ø3	DF <RETURN>	
Ø3Ø4	4C <RETURN>	To return to start
Ø3Ø5	ØØ <RETURN>	
Ø3Ø6	Ø3 <RETURN>	

Then type "." again to get the address mode.

Type

<u>Ø3ØØG</u>

to run the program starting at location 3ØØ hexadecimal.

This program will produce a "saw-tooth" (roughly triangular) waveform at the DAC output. Music generation of pleasing quality, immitative of musical instruments can be played by this device (with additional programming).

You are cautioned that the DAC output should not be tied together with any other output of the computer (such as the tone generator). Further, only one audio output should be used at a time since the register assignment of the audio output devices is the same.

6. Indirect Files

The indirect file is an uncommonly powerful mechanism to manipulate and combine separate programs.

The need for the indirect file arises from two characteristics of our operating system. First, in order to do editing we need to know where a given statement resides in memory. When Assembly language programs are stored, a somewhat compressed form (tokenized) is used to save memory. This makes it difficult to know where a given statement is located in memory. Second, if we wish to load two BASIC programs (assumed to have compatible statement numbers), i.e., you haven't used the same statement numbers in both programs, the operating system would wipe out the first program when it loaded the second.

These potential problems encourage us to place the ASCII coded text sequentially into a single file in memory (similar to a file on disk). Also, it is desirable to be able to keep the two loaded modules (programs) together, contiguous, without garbage in between. The disk file handling routines do not give the fine control that the indirect file does. In an indirect file, we can point to the individual characters in a string of text. For these reasons, indirect file handling has been developed under the OS-65D V3.1 system. The indirect file provides a method of temporarily storing ASCII code.

The indirect file is stored in high memory. The address of the indirect file is stored in 9554 (high byte only). The low half of the indirect file address is assumed to be Ø. For a 24K system, the POKE to store the high address byte is

POKE 9554,80

The high byte of the indirect file address, for different memory configurations is

Memory Size	POKE 9554 with Decimal
24K	80
32K	96
40K	112
48K	128

These suggested memory allocations provide a balance between indirect file size and available user work space. In a 24K system, this allocation of memory allows 4K bytes for the indirect files. Additionally, the indirect file input address must be POKEd at location 9368 with the same table value. For a 24K system this is

POKE 9368,80

First Example: Combining Two Programs

Our goal is to take the first program of two programs and temporarily store it in the indirect file. Then we wish to enter a second program into the BASIC work space, but the LOAD command normally causes overwriting of the first program.

In order to avoid overwriting of one program by another, indirect files allow us to use the steps

1) clean out the work space by typing

   NEW

2) enter a program from the keyboard or a disk file
3) store the newly entered program in an indirect file

4) clear the work space again. This time, we do it only to illustrate that the old program is removed.

5) enter a new program (with statement numbers that do not conflict with the first program).

6) bring the indirect file back into the work space. Now both programs are in the work space and have been merged together.

Let's apply these steps in a short example.

The commands to combine two short programs would be

```
POKE 9554,80 : REM SET INDIRECT FILE OUTPUT FOR 24K SYSTEM

POKE 9368,80 : REM INDIRECT FILE INPUT FOR 24K SYSTEM
```

The first program is then typed

```
10 PRINT"TEST1" : REM SHORT EXAMPLE!
```

The program is transferred to indirect file by typing

<u>LIST</u> <SHIFT K><RETURN>   Note: at the same time pressing <SHIFT K> = [

The listing will appear on the video screen and the program will be transferred to the indirect file in upper memory. We now close the indirect file by typing

SHIFT M <RETURN>   Note: at the same time pressing <SHIFT M> =]

The symbols

]]

will be displayed, along with an error message

?SN ERROR

which should be ignored.

Typing

<u>NEW</u>

will assure ourselves that the program is removed from the BASIC work space.

Now enter the second program

```
20  PRINT"TEST2"
```

The command

```
LIST
```

will assure ourselves that only statement 20 is in the work space. Typing

<CONTROL X>

will transfer the indirect file back into the work space. Either the RUN command or the LIST command shows that both programs are now resident in the BASIC work space.

Our example has been extremely short. You are cautioned that a large program in the BASIC work space could overwrite the indirect file.

<u>Second Example: Creating a Buffer for a Bufferless Program</u>

This example illustrates adding a buffer to a previously written program which lacked a necessary buffer. The original program could be loaded from its file, say FILE1, by

```
DISK!"LO FILE1"
```

Note at this point PEEKs could be done to verify that no buffer was in front of the program, FILE1. Again, we POKE the indirect file I/O addresses for 24K systems

```
POKE 9554,80
POKE 9368,80
```

Typing

<u>LIST</u> < SHIFT K><RETURN>

and

<SHIFT M><RETURN>

writing FILE1 into the indirect file and closes that file. Type

NEW

to remove FILE1 from the BASIC work space.

Run the program "CHANGE" to create the needed buffer. Now, reload FILE1 from the indirect file by typing

<CONTROL X> <RETURN>

The original program with its newly acquired buffer is now resident in the BASIC work space. This program can be stored with the PUT command back on its original disk file (caution, your program is now larger by the buffer size one or two tracks) by

DISK!"PUT FILE1"

This completes the examples. Since the indirect file stores its data as ASCII characters, it may be useful for your file manipulation programs. There is a potential for greater utility than these examples with your application. The indirect ASCII file is a subtle but powerful tool for experienced programmers.

Appendix A

WARRANTY

Ohio Scientific fully-assembled products are covered by a limited warranty. C4P systems are covered for a period of sixty days against defects in materials and workmanship to the extent that any malfunction not caused by abuse, misuse, or mishandling will be repaired or corrected without charge to the owner provided that the unit is returned postpaid to Ohio Scientific within sixty days of day of receipt by the user.

Beyond this sixty day period, up to one year from day of receipt by the user, the system is further warranted against defects in materials to the extent that Ohio Scientific will repair or replace them, charging only for labor on the portion of the electronic component that is manufactured by Ohio Scientific, without charge for the part(s). This warranty includes power supplies and floppy disk drives. It specifically excludes terminals, video monitors, audio cassettes and some keyboards. Ohio Scientific's only obligation under these terms, in either case, is to repair the unit and return it once it has been delivered postpaid to Ohio Scientific. Typical turn-around time under this warranty is two to three weeks plus shipping time from the factory. Ohio Scientific cannot be held responsible for delays beyond its control such as those caused by shipping or long delivery of replacement components, e.g., floppy disk drives, etc.

Ohio Scientific reserves the ultimate authority to determine what constitutes in-warranty repair in circumstances where circuit modification, abuse, misuse, or shipping damage occur. The warranty is also subject to the use of proper packing material in any returns. This is the only warranty expressed or implied by Ohio Scientific and the only warranty which any Ohio Scientific agent is authorized by Ohio Scientific to give in conjunction with the product. Any maintenance or extended warranties that the end user may entertain with an Ohio Scientific representative or dealer are solely between that representative and the customer and are in no way authorized or supported beyond the extent of the above stated warranty by Ohio Scientific. The support of such warranty or maintenance contract is the sole responsibility of the agent offering the warranty.

Ohio Scientific software offers absolutely no warranty. The software is always thoroughly tested and thought to be reasonably bug-free when released. Ohio Scientific maintains a full staff of software experts and will endeavor to correct any serious bugs that may be discovered in the software after release in a reasonable amount of time. However, this is a statement of intent and not a guarantee in such matters.

TROUBLESHOOTING

If you encounter any difficulty in procedures in this manual, first refer to the following troubleshooting guides. If they do not provide sufficient help for you to solve your problems, proceed to the end of this section.

1. Order does not seem complete. First check to see that all packages specified have arrived. Carefully look over the packing lists, manuals, and this manual to determine what is supposed to be present in your system. If you have further doubts, check with the dealer or representative from whom you purchased your system.

2. Unit(s) mechanically damaged in shipment. Report damages or losses immediately to carrier. All units are shipped by Ohio Scientific fully insured. Under no circumstances should you ship the unit back in such condition as it would then be impossible to determine where the unit was damaged. This can cause a long drawn-out dispute with the carrier especially if the unit was transported by different carriers.

3. User has difficulty in following manual because of high level of technology involved. Suggestions: obtain assistance from your local Ohio Scientific dealer or representative. If you ordered factory direct, or are at a considerable distance from the dealer, contact your local hobby club and see if any members can assist you. Hobby club members are generally very willing to help out, which is a major reason they are in the club. Current club activities are listed in BYTE, Kilobaud Microcomputing and Interface Age. Any local computer store should be able to assist you in becoming a computer club member.

4.* Reset light does not illuminate on power-up. Carefully check power connections. Check to see if unit is plugged in, that the power switch is on and that power is present at the power outlet. If so, turn the unit off and unplug it. Check the 2 amp fuse at the back of the unit and check the reset light itself by pulling the lens cap out and making sure that the lamp is properly seated in its socket.

5.* Reset switch is dimly lit or not lit at all after you have checked with the above procedures. Carefully inspect the PC board portion of the computer for foreign matter such as a wire cutting or something leading out from the PC board. Also check to see that all PC boards are properly seated, and that any ribbon cables are properly seated in their sockets. If the unit light is only dimly lit, remove about half of the PC boards. If the light comes up to full brightness with these out, put those boards back in and pull the other ones out. If the same condition occurs, it means that there is a power supply malfunction and that the unit will have to be returned for repair. If the power supply folds back when some PC boards are out, and not with others, you should be able to isolate the board causing the foldback. That board most likely has foreign matter across it, causing the short on the board.

6. Power supplies look fine, but the computer doesn't seem to reset at all or properly. Symptons: nothing comes out on serial terminal or screen doesn't clear on video system. Solution: again, give the system a careful visual inspection. At this point, it

*C8P only.

would be invaluable to have access to another Ohio Scientific computer system by way of a dealer or another computerist. If neither is available, and you do not wish to or are not able to attach the actual circuitry of the system, it will most likely be necessary to return the unit for repair.

7. System works fine in machine code, but in BASIC you consistently receive SN error message (Syntax error). Carefully refer to the example given in the BASIC User's Manual.

In Case of Difficulty

If you encounter a problem with your system, first carefully look over the trouble-shooting hints in your procedures. The great majority of problems encountered on new computers result simply from the user's unfamiliarity with the computer system. If you decide that you cannot resolve the problem yourself, contact the representative or dealer from whom you purchased the computer. Your local OSI dealer representative should be able to help you by providing guidance on operating procedures, and in the case of an actual computer malfunction, should be able to substitute PC boards and subassemblies to isolate the problem. He should then also provide the service of getting the replacement or repair for the malfunctioning unit.

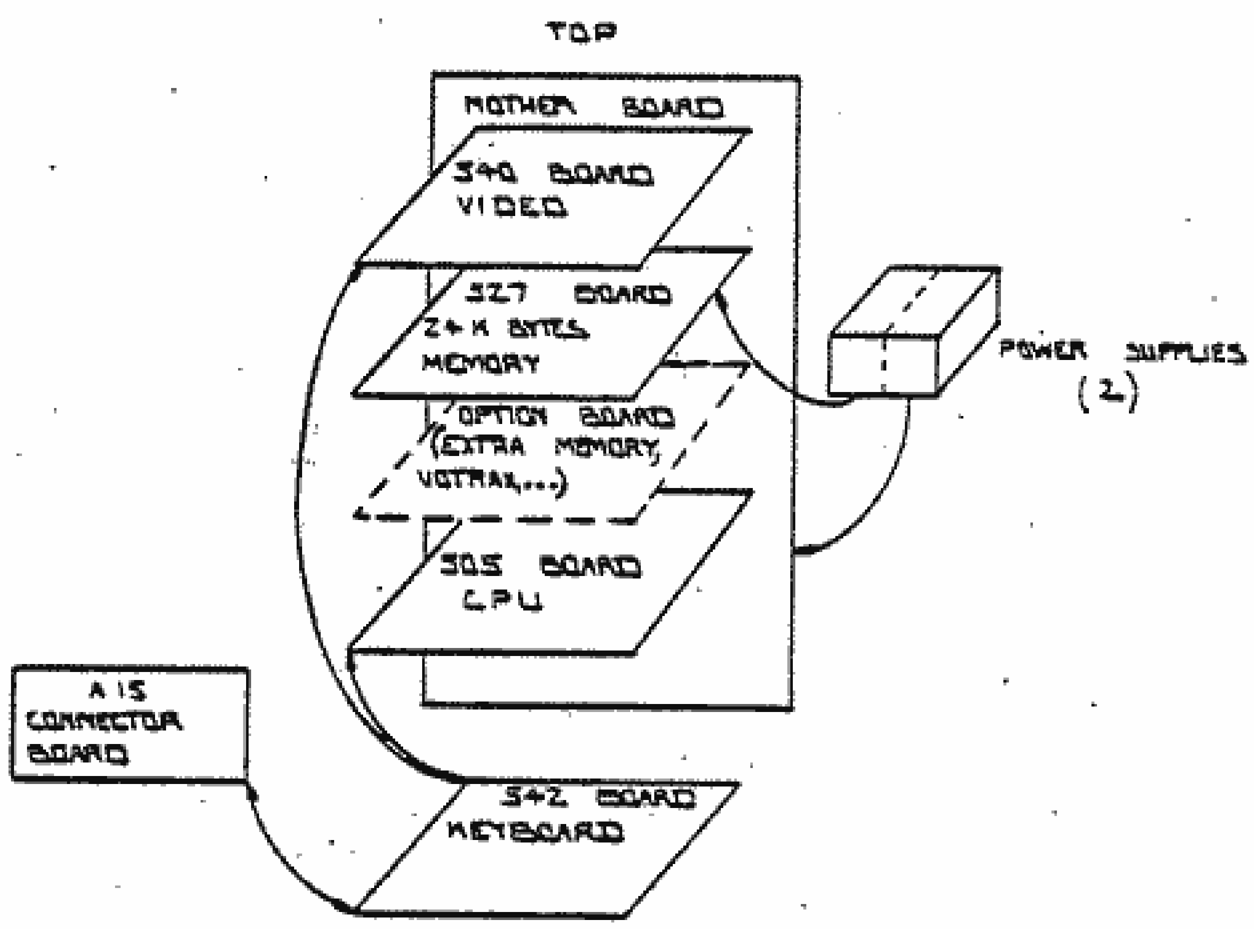
COLOR TUNING (Hetrodyning adjustment)

If color has been selected and does not appear or if a "barber pole" effect is seen at color boundaries, a simple operator adjustment will correct these problems.

The C4P with color option has crystal oscillators to set the rate of display of the image and the color information. A shaft on a potentiometer (see Figure 1) provides adjustment of the relative rates of these oscillators. Normally, adjustment is made after the circuits have warmed up for half an hour. Additional adjustment should not be necessary once the computer has warmed up.

The Machine Organization

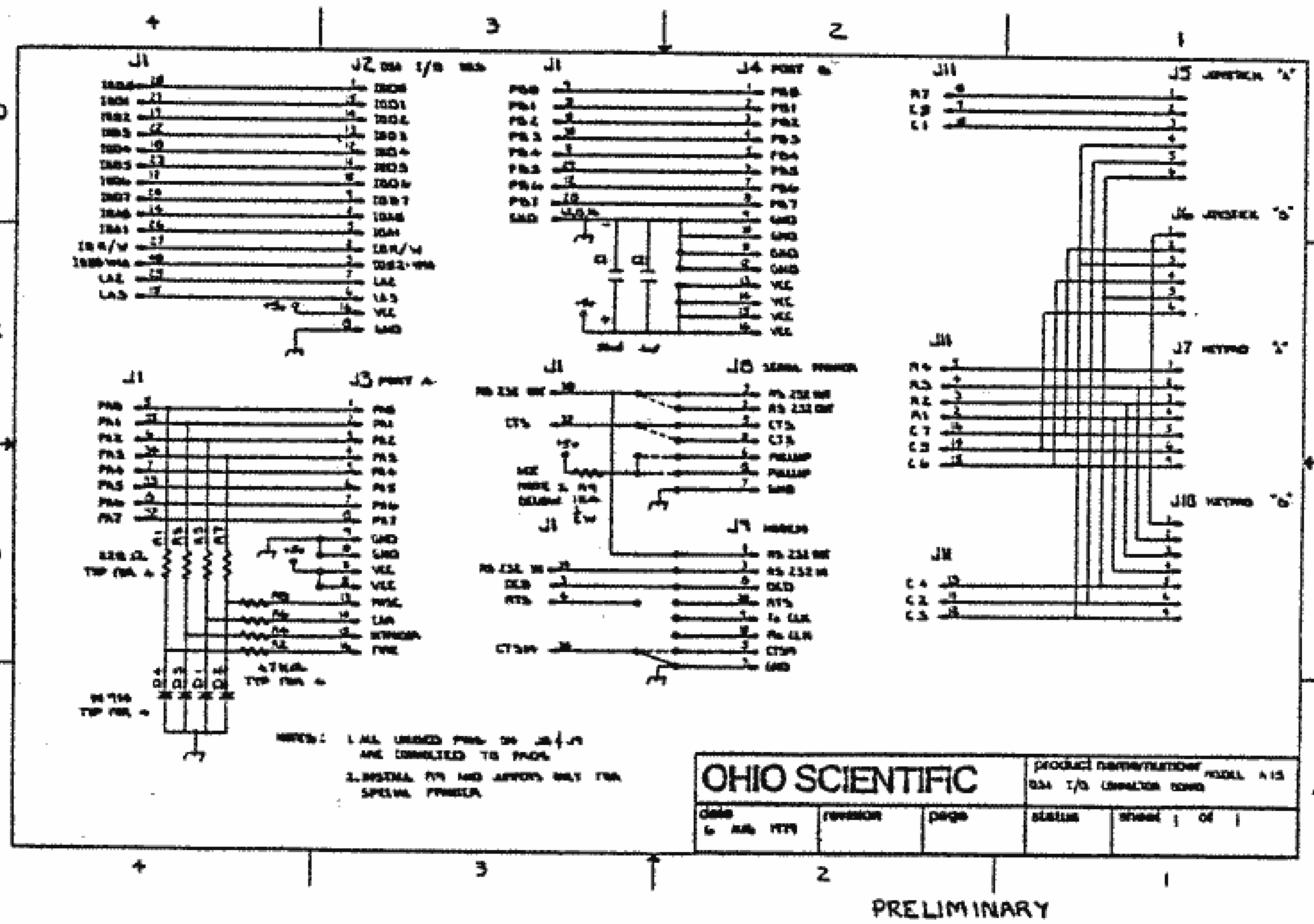
The high density and modularity of your C4P system is defined by the board structure.



This system permits economical extensions of your system as your computing demands increase.

Detailed Pin Connections

The connectors shown on the A-15 board have the pin connections detailed below. Reference to schematic information accompanying equipment is advised if more extensive use than the manual examples is anticipated. Nomenclature is specified in the schematic diagrams. This listing is intended to provide pin outs of the PIA's and the printer/modem in support of the manual examples, only.



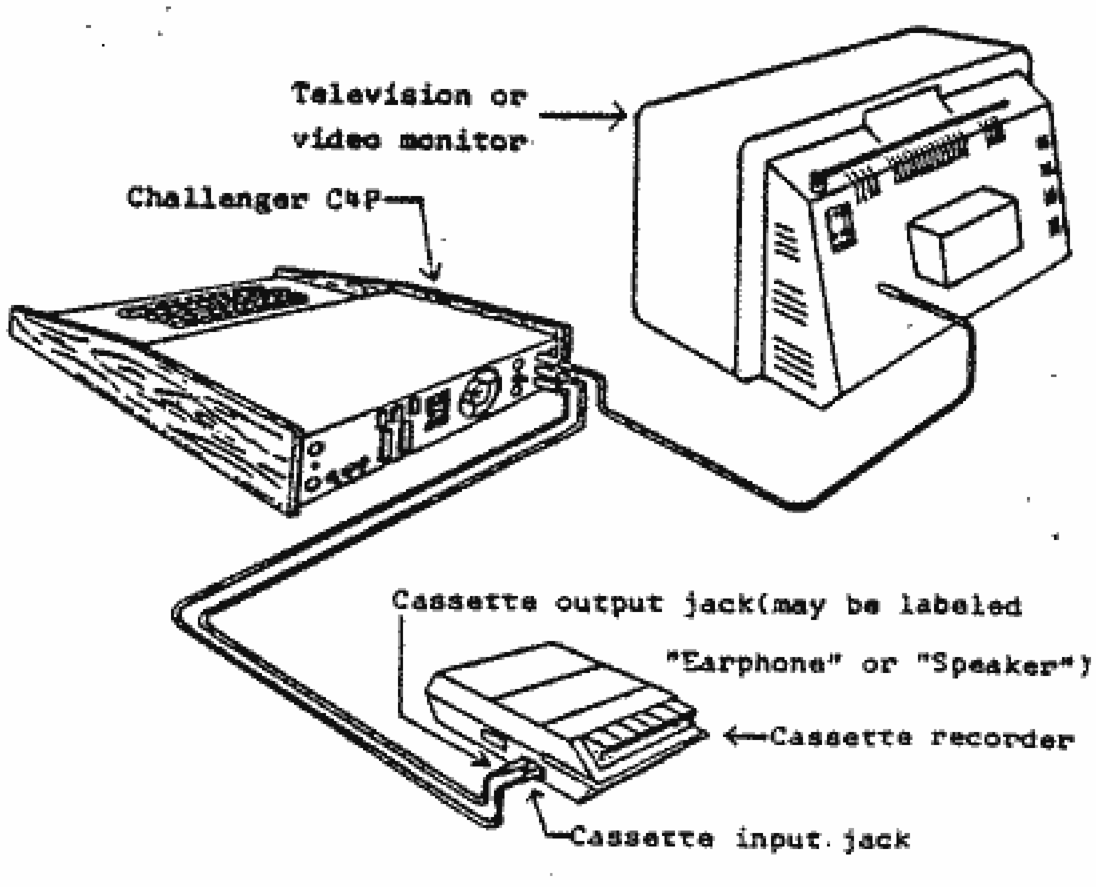
Appendix B

Cassette based C4P Directions

The manual to this point, has assumed the reader is a C4P MF user. The mini-floppy disk provides a large performance benefit for the relatively small investment above a C4P (cassette) system; the chief benefits of the C4P MF are file handling and high speed data transfer. The cassette provides an economical bulk storage medium, though the data transfer rate is considerably lower than disk's rate.

If you have read this far, you have probably opted for cassette. The internal configuration of computer components is slightly different than the mini-floppy configuration. Externally, the computer and accessories should agree with Figure

The cassette recorder should be a medium price audio tape recorder. If price is indicative of quality, then $35-$50 would be a price guide. Volume and tone controls should be set at mid-range. If you do not use 110V AC for the recorder power, be sure to use fresh batteries. (Speed variations due to weak batteries can create errors.)



Computer Set-Up (Cassette systems)

The precautions and discussion given in the main part of this manual for the C4P MF (mini-floppy) system, still apply. As a reminder these are outlined.

1. Assemble the computer system according to Figure Using OSI supplied cables will assure reliable and firm connections between units.

2. Turn on the computer. The switch is on the back panel.

3. Turn on the monitor. (Only OSI modified monitors or RF modulators should be used. Damage produced by unauthorized monitors will void all warranty coverage.)

4. Turn on the cassette recorder power.

5. Press the "BREAK" key.

6. Rewind the cassette so that the tape "leader" is visible on the take-up spool. OSI software will be supplied on high quality tapes. Use of low quality tapes will cause erratic performance and excessive recorder wear.

7. Respond to the terminal screen message

   C/W/M?

   by pressing the "SHIFT LOCK" key down and then respond

   C <RETURN>

   If the "SHIFT LOCK" key is not depressed, the keyboard message will not be understood by the computer.

8. When the computer requests

   MEMORY SIZE?

   just press the "RETURN" key.

9. The computer will next ask

   TERMINAL WIDTH?

   Again, press the "RETURN" key

10. The prompt

   OK

   should appear at the bottom of the screen. If it does not, repeat steps 1 to 1Ø again.

You are now in the BASIC program. The cassette supported C2P is a BASIC-in-ROM system, having a 6-digit BASIC stored in read only memory (ROM). Two small but readily apparent differences between BASIC-in-ROM and disk based BASIC are

1. The character delete symbol <SHIFT O> will print an underline symbol rather than erase the deleted character. The statement

   ```
   10 PRX _INT "HELP"
   ```

   would appear as

   ```
   10 PRINT"HELP"
   ```

   if we did a

   ```
   LIST 10
   ```

   command, showing the symbol X has been truly deleted.

2. Error message codes will appear different than the list given for disk based BASIC. A list of error codes for both versions of BASIC is given in Table A.5.

The cassette based systems are not permitted to use back panel connections J2 to J4 and J8 and J9.

To LOAD Cassette Programs into RAM (memory)

Enter the BASIC program, as shown in the previous section.

1. Place the demonstration cassette in the recorder.

2. Rewind the cassette tape. When the tape stops rewinding return the selection switch(s) to STOP.

3. Type

   NEW <RETURN>

   This will clear memory in preparation for reading the cassette.

4. Type

   LOAD

   <u>but not</u> RETURN

5. Start the cassette in the PLAY mode, in order to play back the demonstration programs into the computer memory.

6. As soon as the tape leader has moved past the recorder head (is no longer visible on the wound up reel), press the

   <RETURN>

7. The computer will type

   ?S┘ ERROR

   OK

   Which you may ignore. The computer will then list the program being read. The program appears on the terminal screen and is simultaneously stored in memory. If you have a large unused tape region between the tape leader and your program, meaningless characters will be printed. They may be ignored, as they will not affect the program operation.

8. When the program is finished listing, you will see printed

   OK

   ?S┘ ERROR

   OK

9. Turn off the cassette recorder, then type

   <SPACE>

   then

   <RETURN>

   Your program is now in memory. You may examine the program by typing

   LIST <RETURN>

10. When finished, store the cassette away from heat or magnets. Do not leave the cassettes on the computer case, as the temperature and proximity to the iron transformers can degrade the programs stored on tape.

SAVING Programs on Cassette

Let's start by clearing memory by typing

NEW <RETURN>

The computer responds

OK

and writing a short program

```
10  PRINT"NOW IS THE TIME"
20  PRINT"FOR ALL GOOD MEN"
30  END
```

which we wish to store on tape.

1. Rewind the tape.
2. Type

   SAVE <RETURN>

   The computer responds

   OK
3. Now type

   LIST

   but not RETURN !
4. Start the recorder in the record mode. This operation is obtained by pressing the RECORD and PLAY switches, simultaneously. (This two switch operation is meant to reduce inadvertent writing over programs we did not want destroyed.)
5. As soon as the leader passes the recording heads (disappears from sight on the windup reel), type

   <RETURN>
6. When the listing is complete, turn off the tape recorder and type

   LOAD <RETURN>

   <SPACE>

   <RETURN>

7. You may now rewind the tape and check that your recording is satisfactory by following the instructions to LOAD the cassette.

<u>Use of Cassettes as a Data Storage Medium</u>

Intermediate data within programs can be stored on cassette. This provides easy retrieval of data and intermediate calculations for future use.

As an example, let's print the numbers 1 to 15 on the cassette. After rewinding the tape, the sequence of operations would be

1. Write the program to create the desired data, such as

```
10  FOR I=1 TO 15
20  PRINT I
30  NEXT I
40  END
```

2. Type

   <u>SAVE</u> <RETURN>

3. Type

   <u>RUN</u>

   <u>but not</u> <RETURN>

4. Start the recorder in the record mode (PLAY and RECORD switches depressed). As soon as the tape leader has passed the recording head, press

   <RETURN>

5. The data will be recorded on tape and listed on the terminal screen.

6. When the listing of data is complete, turn off the tape recorder and type

   <u>LOAD</u> <RETURN>

   <SPACE>

   <RETURN>

   to return to normal operation.

You will note that this set of procedure steps was almost the same set we had used to SAVE a program.

We can use this data as input for another program in a similar manner.

Reading Data From Cassette Tape

In a manner similar to LOADing programs from cassette, we can read data from cassette. The steps are

1. Rewind the cassette tape.
2. Type

   NEW <RETURN>

3. Enter your program which will use the data on tape. A typical program might be

```
10  INPUT A
20  PRINT "DATA IS=";A
30  IF A 15 THEN GOTO 10
40  END
```

   Now type

   RUN

   but not <RETURN>

4. Start the tape in the PLAY mode to play back the data. When the tape leader is beyond the recorder's head, then press

   <RETURN>

5. The requests for data will be shown on the terminal screen as typically

```
?1          { DATA FROM TAPE
DATA IS=1   { ANSWER FROM PROGRAM
?2
DATA IS=2
```

   etc.

6. Upon completion of the program (or the tape being wound up on the reel), turn off the tape recorder. Then type

   <SPACE>

   and

   <RETURN>

   You will now be in the BASIC program.

These techniques should permit a flexible use of your cassette, both as a program and data storage medium.

For extensive data handling, however, the drive control of a disk will give enhanced speed and control. Therefore, its use is encouraged.

BASIC-in-ROM Error Messages

CODE		DEFINITION
DD	D	Double Dimension: Variable dimensioned twice. Remember subscripted variables default to dimension 1ø.
FC	F	Function Call error: Parameter passed to function out of range.
ID	I	Illegal Direct: Input or DEFIN statements can not be used in direct mode.
NF	N	NEXT without FOR:
OD	O	Out of Data: More reads than DATA
OM	O	Out of Memory: Program too big or too many GOSUBs, FOR NEXT loops or variables
OV	O	Overflow: Result of calculation too large for BASIC.
SN	S	Syntax error: Typo, etc.
RG	R	RETURN without GOSUB
US	U	Undefined Statement: Attempt to jump to non-existent line number
/ø	/	Division by Zero
CN	C	Continue errors: attempt to inappropriately continue from BREAK or STOP
LS	L	Long String: String longer than 255 characters
OS	O	Out of String Space: Same as OM
ST	S	String Temporaries: String expression too complex.
TM	T	Type Mismatch: String variable mismatched to numeric variable
UF	U	Undefined Function

Appendix C

Memory Map (RAM)

Within a computer, different programs and programmers will lay claim to memory locations. Though these locations are not needed by all programs, prudence would encourage us to make a list of all the locations we know that have been committed to different operating systems and utility programs. If we avoid using these locations, we minimize the risk of a program failing for unexplained reasons. The reason is generally that a value needed by a system program was found destroyed by a user program.

Also, knowing the reserved locations permits us to take advantage of these locations. For example, the memory which is dedicated to screen display could be used as extra storage (though it messes up the display by doing this). (Also, we can read values off the screen by looking into the memory location corresponding to the screen position.)

Though you can program well without needing this map, the preceding justification merits this list.

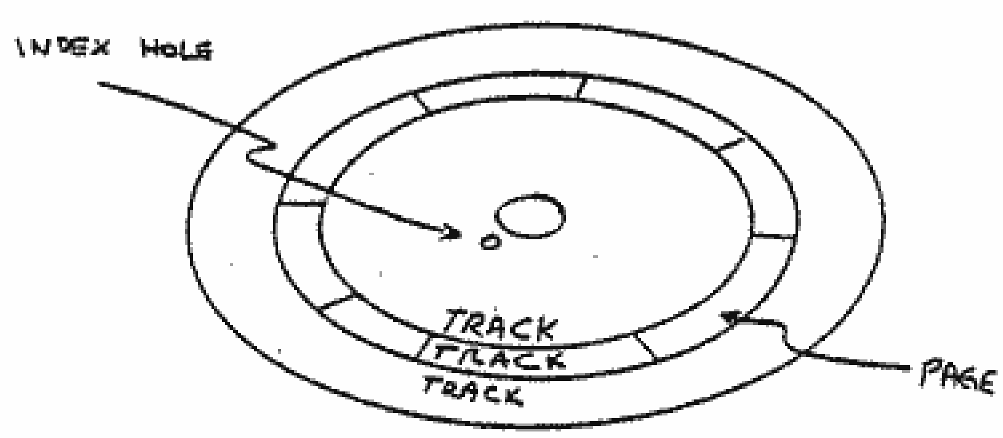
C4P Memory

Decimal Location	Hexadecimal Location	Use
Ø000 – Ø255	Ø000 – ØØFF	65Ø2 Page Zero
Ø256 – Ø511	Ø1ØØ – Ø1FF	65Ø2 Stack (Page 1)
Ø512 – 8959	Ø2ØØ – 22FF	Transient program area for user's language processor
896Ø – 9819	23ØØ – 2658	I/O Handlers
982Ø – 1Ø826	265C – 2A4A	Floppy Drivers
1Ø827 – 11896	2A4B – 2E78	Disk Operating System (DOS)
11897 – 12664	2E79 – 3178	Page Ø1/1 Swap Buffer
12665 – 1292Ø	3179 – 3278	DOS Extensions
12921 – 12925	3279 – 327D	Source file header information
12926 – TO END OF MEMORY	327E	Source File

Mini-Floppy Disk Organization

It is usefol to know how information is placed on th disk, in order to plan your use of the disk.

Each mini-floppy is organized into 40 tracks, numbered from 0 through 39. Track 0 is near the outside edge of the disk while track 39 is close to the center. All tracks are circular tracks similar to the tracks on a phonograph record. See the diagram below



Each track may be subdivided into sectors and pages. A page is a block of 256 bytes while a sector must be an integer multiple of pages (up to 8 pages, of course). BASIC programs are limited to integral multiples of tracks (2tracks not 1½ pages) but machine code programs may be in sectors of variable page lengths. Several machine code routines (of various or similar sizes) may be saved on one track in this manner

For example, the disk directory found elsewhere in this section shows that tracks 6,9,11 and 12 contain various combinations of machine code programs in segments. Specifically, track 12 has four one page sectors. One should note that the BASIC program BEXEC* on track 14 comprises one 8 page sector.

Osi software utilizes single sided, single density soft-sectored disks. Soft-sectored disks have one index hole which provides a timing reference for hardware purposes.

When we store information on the disk, we usually assign the file of information a "file name". File names are constrained to 6 or fewer characters, the first character being a letter.

Certain tracks are dedicated to the disk operating system, as shown in the table below.

TRACK	USE
Ø	DOS-part 1
1	DOS-part 2
2-6	9½ Digit BASIC
7-9	Assembler/Editor (ASM)
10-11	Extended Monitor (EM)
12	Sector 1 - Directory Page 1
12	Sector 2 - Directory Page 2
12	Sector 3 - BASIC Overlays
12	Sector 4 - GET/PUT Overlays
13	COPIER/TRACKØ Utility
14-39	User and/or utility programs

When a new disk is placed in operation, it is initialized to place timing marks on the disk and check disk quality. If you wish to clean a file of a disk which is in service (in contrast to cleaning the entire disk), the "ZERO" program provides this service.

The disk directory, whose entries are made by the CREATE program, does the bookkeeping of placing file names into the directory. By keeping the directory up to date, efficient use of this bulk storage medium can be enjoyed.

DISK DIRECTORY

MINI-FLOPPY 5¼ INCH DISK

PAGE________
DATE________

#1 Program	#2 Track	#3 Sector or Format	#4 Start of Transfer	#5 Length in Pages	#6 Go Address	#7 Comments
OS-65D V3.0 and V3.1 Part I	Ø	1	22ØØ	8		1st page overlaid by T6 & T11
" " " " Part II	1	1	2AØØ	8		(T means track)
BASIC Part I	2	1	Ø2ØØ	8		
" Part II	3	1	ØAØØ	8		
" Part III	4	1	12ØØ	8		
" Part IV	5	1	1AØØ	8		2ØC4-21C3 overlaid by T 12,3
" Part V	6	1	22ØØ	1		
Assembler Part I	7	1	Ø2ØØ	8		
" Part II	8	1	ØAØØ	8		
" Part III	9	1	12ØØ	5		
EM Part I	1Ø	1	17ØØ	8		
" Part II	11	1	1FØØ	4		
Directory Page I	12	1	2E79	1		Overlaid by T 12, 4 on OPEN
" Page II	12	2	2E79	1		" " " " " " "
BASIC Overlays	12	3	2ØC4	1		
PUT/GET Overlay	12	4	2E79	1		
COPIER/TRACK Ø Utility	13	1	Ø2ØØ	5		
BEXEC*	14	1	327F	8		
COMPAR	39	1	Ø2ØØ	5	Ø2ØØ	Not present on all disks
		2	2ØØØ	2		

Disk BASIC: Statements

In the following examples

V or W is a numeric variable, X is a numeric expression,

X$ is a string expression, I or J is a truncated integer.

NAME	EXAMPLE	COMMENTS
INPUT	1Ø INPUT A	Variable A will be accepted from the terminal. A carriage return will terminate input.
DEF	1Ø DEF FNA (V)=V*B	User defined function of one argument.
DIM	11Ø DIM A (12)	Allocates space for Matrices and sets all matrix variables to zero. Non-dimensioned variables default to 1Ø.
END	999 END	Terminates program (optional).
FOR,NEXT	1Ø FOR x=.1 to 1Ø STEP.1 2Ø 3Ø NEXT X	STEP is needed only if X is not incremented by 1. NEXT X is needed only if FOR NEXT loops are nested, if not, NEXT alone can be used (variables and functions can be used in FOR statements).
GOTO	5Ø GOTO 1ØØ	JUMPS to line 1ØØ
GOSUB, RETURN	1ØØ GOSUB 5ØØ 5ØØ . . . . 6ØØ RETURN	Goes to subroutine, RETURN goes back to next line number after the GOSUB.
IF...THEN	1Ø IF X=5 THEN 5 1Ø IF x=5 THEN PRINT X 1Ø IF X=5 THEN PRINT X:Y=Z	Standard IF-THEN conditional with the option to do multiple statements.
IF...GOTO	1Ø IF X=5 GOTO 5	Same as IF-THEN with line number.
ON...GOTO	1ØØ ON I GOTO 1Ø,2Ø,3Ø	Computed GOTO If I=1 then 1Ø If I=2 then 2Ø If I=3 then 3Ø
DATA	1Ø DATA 1,3,7	Data for READ statements must be in order to be read. Strings may be read in DATA statements.

PRINT	1Ø PRINT X 2Ø PRINT "Test"	Prints value of expression Standard BASIC syntax with ,;" formats.
READ	49Ø READ V, W	Reads data consecutively from DATA statements in program.
REM	1Ø REM	This is a comment for non-executed comments.
RESTORE	5ØØ RESTORE	Restores initial values of all data statements.
STOP	1ØØ STOP	Stops program execution, reports a BREAK. Program can be restarted via CONT.

Disk BASIC Functions

Function	Comment
ABS (X)	For X≥Ø ABS(X)=X For X<Ø ABS(X) =-X
INT (X)	INT (X) = largest integer less than X
RND (X)	RND (Ø) generates the same number always. RND (X) with the same X always generates the same sequence of random numbers NOTE: [(B-A)*RND (1)+A] generates a random number between B and A.
SGN (X)	If X>Ø then SGN(X)=1 If X=Ø then SGN(X)=Ø If X<Ø then SGN(X)=-1
SIN (X)	Sine of X where X is in radians.
COS (X)	Same for COS, TAN, and ATN (ARC TAN).
TAN (X)	
ATN (X)	
SQR (X)	Square root of X.
TAB (I)	Spaces the print head I spaces.
USR (I) See I/O section	
EXP (X)	e^X where e=2.71828.

FRE (X)	Gives number of Bytes left in the work space
LOG (X)	Natural log of X. To obtail common logs use Common log(x)=LOG(x)/LOG(10).
POS (I)	Gives current location of terminal print head.
SPC (I)	Prints I spaces, can only be used in print statements.

STRINGS

Strings can be from Ø to 255 characters long. All string variables end in $, such as A$, B9$, and HELLO$.

Disk BASIC String Functions

ASC (X$)	Returns ASCII value of first character in string X$.
CHR$ (I)	Returns an I character string equivalent the ASCII value above.
LEFT$ (X$,I)	Gives left most I characters of string X$.
RIGHT$(X$,I)	Gives right most I character of string X$.
MID$ (X$,I,J)	Gives string subset of string X$ starting at Ith character for J characters. If J is omitted, goes to end of string.
LEN (X$)	Gives length of string in bytes.
STR$ (X)	Gives a string which is the character representation of the numeric expression of X. Example X=3.1 X$=STR$(X) X$="3.1"
VAL (X$)	Returns string variable converted to number. Opposite of STR$(X).

Disk BASIC Commands

NAME	EXAMPLE	COMMENTS
LIST	LIST LIST 100	Lists program Lists program from line 100. Control C stops program listing at the end of current line.
NULL	NULL 3	Inserts 3 nulls at the start of each line to eliminate carriage return bounce problems. Null should be 0 when entering paper tapes from Teletype readers. When punching tapes NULL = 3. Higher settings are required on faster mechanical terminals.
RUN	RUN	Starts program execution at first line. All variables are reset. Use an immediate GOTO to start execution at a desired line.
	RUN 200	GOTO 200 with variables reset.
NEW	NEW	Deletes current program.
CONT	CONT	Continues program after Control C or STOP if the program has not been modified. For instance a STOP followed by manually printing out variables and then a CONT is a useful procedure in program debugging.
LOAD	LOAD	Used in cassette and Disk BASIC only.

Disk BASIC Operators

SYMBOL	EXAMPLE	COMMENTS
=	A=10 LET B=10	LET is optional
-	-B	Negation
<SHIFT N>	X^4	X to the 4th power
		(C^D with C negative and D not an integer gives an FC error.)
*	C=A*B	Multiplication

/	D=L/M	Division
+	Z=L+M	Addition
-	J=255.1-X	Subtraction
<>	10 IF A<>B THEN 5	Not Equal
>	B>A	B greater than A
<	B<A	B less than A
<=,=<	B<=A	B less than or equal to A
=>, =>	B=>A	B greater than or equal to A
AND	IF B>A AND A>C THEN 7	If <u>both</u> expressions are true then 7.
OR	IF B>A OR A>C THEN 7	If <u>either</u> expression is true then 7.
NOT	IF NOT B<>X THEN 7	If B =A then 7.

AND, OR, and NOT can also be used in Bit manipulation mode for performing Boolean operations of 16 bit 2s complement numbers (-32768 to +32767)

<u>EXAMPLES</u>

<u>EXPRESSION</u>	<u>RESULT</u>
63 AND 16	16
-1 AND 8	8
4 OR 2	6
10 OR 10	10
NOT 0	-1
NOT 1	-2

OPERATOR EVALUATION RULES: Math statements evaluated from left to right with * and / evaluated before + and -. Parentheses explicitly determine order of evaluation.

Precedence for evaluation

1) By parentheses

2) ^

3) Negation

4) * /

5) + -

6) =,<>,<,>,<=,>=

7) NOT

8) AND

9) OR

Disk BASIC--Error Listing

Errors can arise in several contexts. Errors in the BASIC program will be indicated by a two letter mnemonic code. The code and its interpretation are:

ERROR CODE	MEANING
BS	Bad Subscript: Matrix outside DIM statement range, etc.
DD	Double Dimension: Variable dimensioned twice. Remember subscripted variabled default to dimension 1Ø.
FC	Function Call error: Parameter passed to function out of range
ID	Illegal Direct: Input or DEFIN statements can not be used in direct mode.
NF	NEXT without FOR:
OD	Out of Data: More reads than DATA
OM	Out of Memory: Program too big or too many GOSUBs, FOR NEXT loops or variables.
OV	Overflow: Result of calculation too large for BASIC
SN	Syntax error: Type, etc.
RG	RETURN without GOSUB.
US	Undefined Statement: Attempt to jump to non-existent line number.
/Ø	Division by Zero
CN	Continue errors: Attempt to inappropriately continue from BREAK or STOP.
LS	Long String: String longer than 255 characters
OS	Out of String Space: Same as OM
ST	String Temporaries: String expression too complex.
TM	Type Mismatch: String variable mismatched to numeric variable.
UF	Undefined Function.

DOS Error Messages

CODE	MEANING
1	Cannot read sector (parity error)
2	Cannot write sector (reread error)
3	Track zero write protected against that operation
4	Disk is write protected
5	Seek error (track header does not match track)
6	Drive not ready
7	Syntax error in command line
8	Bad track number
9	Cannot find track header within one rev of disk
A	Cannot find sector before one requested
B	Bad sector length value
C	Cannot find file name in directory
D	Read/Write attempted past end of named file

Converting Other BASICS To Run On OSI 6502 BASIC

Strings:

OTHER	OSI
DIM A$ (I,J)	DIM A$(J)
A$ (I)	MID$ (A$,I,1)
A$ (I,J)	MID$ (A$,I,J-I+1)

Multiple assignments: B=C=0 must be rewritten as B=0:C=0. Some BASICS use \ to delimit multiple statements per line. Use ":". Some BASICS have MAT (Matrix Operation) functions which will have to be rewritten with FOR NEXT loops.

Appendix E

POKEs and PEEKs

The following features of OSI BASIC are useful for several applications. The user should be extremely careful with these statements and functions since they manipulate the memory of the computer directly. An improper operation with any of these commands can cause a system crash, wiping out BASIC and the user's program.

STATEMENT/FUNCTION	COMMENT
PEEK (I)	Returns the decimal value of the specified memory or I/O location. (Decimal) Example: X=PEEK (64256) Loads variable X with the 430 board's A/D converter output. (FB00 hex)
POKE I,J	Loads memory location I (decimal) with J (decimal). I must be between 0 and 65536 and J must be between 0 and 255. Example: 10 POKE 64256, 255 loads FB00 with FF (hex).

Useful BASIC POKEs

As systems develop, different locations are committed to hold parameters. Many of these parameters have been mentioned in the text material. These parameters are collected here, along with some other useful parameters which may be needed by an advanced programmer. Some parameters appear several times, since they are relabeled by other utility programs.

Caution, care must be taken when POKEing any of these locations to avoid system errors and subsequent software losses.

Location Decimal	Location Hex	Normal Contents	Use
23	17	132	Terminal width (number of printer characters per line). The default value is 132. Note, this is not to be confused with the video display width (64 characters).
24	18	112	Number of characters in BASIC's 14 character fields (112 characters = 8 fields) when outputting variables separated by commas.
12Ø 121	78 79	127 5Ø	Lo-Hi byte address of the beginning of BASIC work space (note 127=$7F, 5Ø=$32).
132 133	84 85	* *	Lo-Hi byte address of the end of the BASIC work space. (*contents vary according to memory size such as 255($FF) and 95($5F) or $5FFF=24575 or 24K)
222	DE	Ø	Location to enable or disable RTMON (real time monitor). 1 enables and Ø disables RTMON.
223	DF	Ø	Location to start count down timer. A 1 starts the timer, and a Ø stops it.
224	EØ	Ø	Contains the number of hours for timer to count down.
225	E1	Ø	Contains the number of minutes to count down.
226	E2	Ø	Contains the number of seconds to count down.

23Ø-241	E6-F1	Ø	Identifies the I/O masks used for external polling of AC events, i.e. determines which PIA lines are scanned.
249	F9	Ø	Should contain the latest value at 56832 ($DEØØ) and can be PEEKed unlike $DØØ which is a "write only" register. This location does not change the display format but acts to maintain the format during ACTL use.
548 549	Ø224 Ø225	- -	Hi-Lo byte address for AC driver, with no buffers these locations (with AC enabled) will contain $327F.
741	2E5	1Ø	Control location for "LIST". Enable with a 76, disable with a 1Ø.
75Ø	2EE	1Ø	Control location for "NEW". Enable with a 78, disable with a 1Ø.
1797	7Ø5	32	Controls line number listing of BASIC programs, enable with a 32, defeat with a 44.
2Ø73	819	173	"CONTROL C" termination of BASIC programs. Enable with 173, disable with 96.
22ØØ	898	-	The monitor ROM directs mask Ø to load here at $22ØØ.
2888	B48	27	A 27 present here allows any null input (carriage return only) to force into immediate jumping out of the program. Disable this with a Ø. This location overrides 2893 and 2894.
2893 2894	B4D B9C	55 Ø8	Alternate "break on null input" enable/disable location. A null input will produce a "REDO FROM START" message when 2893 and 2894 are POKEd with 28 and 11 respectively.
2972	B9C	58	Normally a comma is a string input termination. This may be disabled with a 13 (see 2976).
2976	BAØ	44	A colon is also a string input terminator, this is disabled with a 13 (see 2972).
87Ø8	22Ø4	41	Output flag for peripheral devices (see peripheral section).
89Ø2	22C6	"	Determines which registers (less 1) RTMON scans (see the AC control section).

8909 8910	22CD 22CE	04	Hi-Lo byte location of PIA for RTMON scanning (see the AC control section).
8917	22DS	-	USR(X) operation code (refer to USR(X) under advanced topics).
8944	22F0	-	Output flag (refer to peripheral section).
8954	22FA	20	Location of JSR to disk USR(X) routine.
8955 8956	22FB 22FC	208 79	Lo-Hi byte address of USR(X) pointer (refer to USR(X) under advanced topics).
8960	2300	*	Memory (RAM) page count minus 1 (where a page = 256 bytes). *Contents depend upon your machine.
8993	2321	02	I/O distributor input flag. See peripheral section.
8994	2322	02	I/O distributor output flag. See peripheral section.
8996	2324	-	Location of random seed for RND function.
8998 8999	2326 2327		Lo-Hi byte address of the pointer to disk buffer 1.
9000 9001	2328 2329		Lo-Hi byte address of the end (plus 1) of the disk buffer area.
9666	25C2	0	When POKEd with N (0-63) and a LIST command is given, this will move the left hand margin to the right N spaces (dashes will echo on the left unless the cursor is removed).
9667	25C3	215	When POKEd with N (207-215) and a LIST command is given this will move the scroll up 4*(215-N) lines.
9680	25d0	95	This location contains the cursor character designation.

USR

The USR function allows linkage to machine language routines such as ultra-fast device handlers, etc.

It is used as

```
X=USR(X)
```

USR(X) FOR COLOR BACKGROUND ----

This is a BASIC program that sets up an ASSEMBLER subroutine under the USR(X) function. The subroutine changes the background color of the entire screen. Note, if a disk system is not used then the BASIC code; DISK!"CA 4FD0=36,1"; must be removed from the program.

To save the assembler program (created by this BASIC program) on disk, type DISK!"SA 36,1=4FD0/1" after running the program. This will let you call the program from disk in any other BASIC program by the command DISK!"CA 4FD0=36,1" instead of running this BASIC code.

Use the following code in BASIC (after the assembler program is loaded into memory) to exicute the assembler routine. NOTE: this must be done after the subroutine is in memory.

```
POKE8955,208:POKE8956,79
```

This is the high and low addresses to tell the computer where the USR(X) function is located in memory.

```
POKE20433,(your color choice, 0-16)
```

This is your choice of color background.

```
X=USR(X)
```

This is the BASIC command for jumping to an assembler subroutine specified by the previous POKEs.

```
100 FORI=20432TO20473:READX:POKEI,X:NEXT
200 DATA162,14,169,0,141,242,79,169,224,141,243,79,173,242,79
210 DATA24,105,1,141,242,79,173,243,79,105,0,141,243,79,201,232
220 DATA240,6,142,0,224,76,220,79,96,0,2
```

For more detail see page 126.

C4P Diskette Directory

ED1

1. Mathink
2. Math Blitz
3. Spelling Quiz
4. Counter
5. Hangman
6. Geography Quiz
7. Definite Integral
8. Add Game

ED2

1. BASIC Tutor I
2. BASIC Tutor II
3. BASIC Tutor III
4. BASIC Tutor IV
5. BASIC Tutor V
6. BASIC Tutor VI
7. Trig Tutor

ED3

1. Trig Tutor
2. Presidents Quiz
3. Homonym Quiz
4. Continents Quiz
5. Base Conversions
6. Math Intro
7. Solar System Quiz

BD1

1. Ratio Analysis I
2. Ratio Analysis II
3. Bond Evaluation
4. Break Even Analysis
5. Bar Graph
6. Trend Line
7. Interest on Loans

BD2

1. Address Book
2. Inventory Demo
3. Mailing List
4. Advertisement Demo
5. Word Processor

PD1

1. Checking Account
2. Savings Account
3. Annuity I
4. Annuity II
5. Biorhythm
6. Calorie Counter
7. Rate of Return

PD2

1. Definite Integral
2. Base Conversions
3. Trend Line
4. Powers
5. Electronics Equation
6. Math Library

GD1

1. Star Wars
2. Space War
3. Hectic
4. Bomber
5. Torpedo
6. Breakout

GD2

1. Etch-A-Sketch
2. Racer
3. Destroyer
4. Lander
5. Hide & Seek
6. Bomber
7. Tiger Tank

GD3

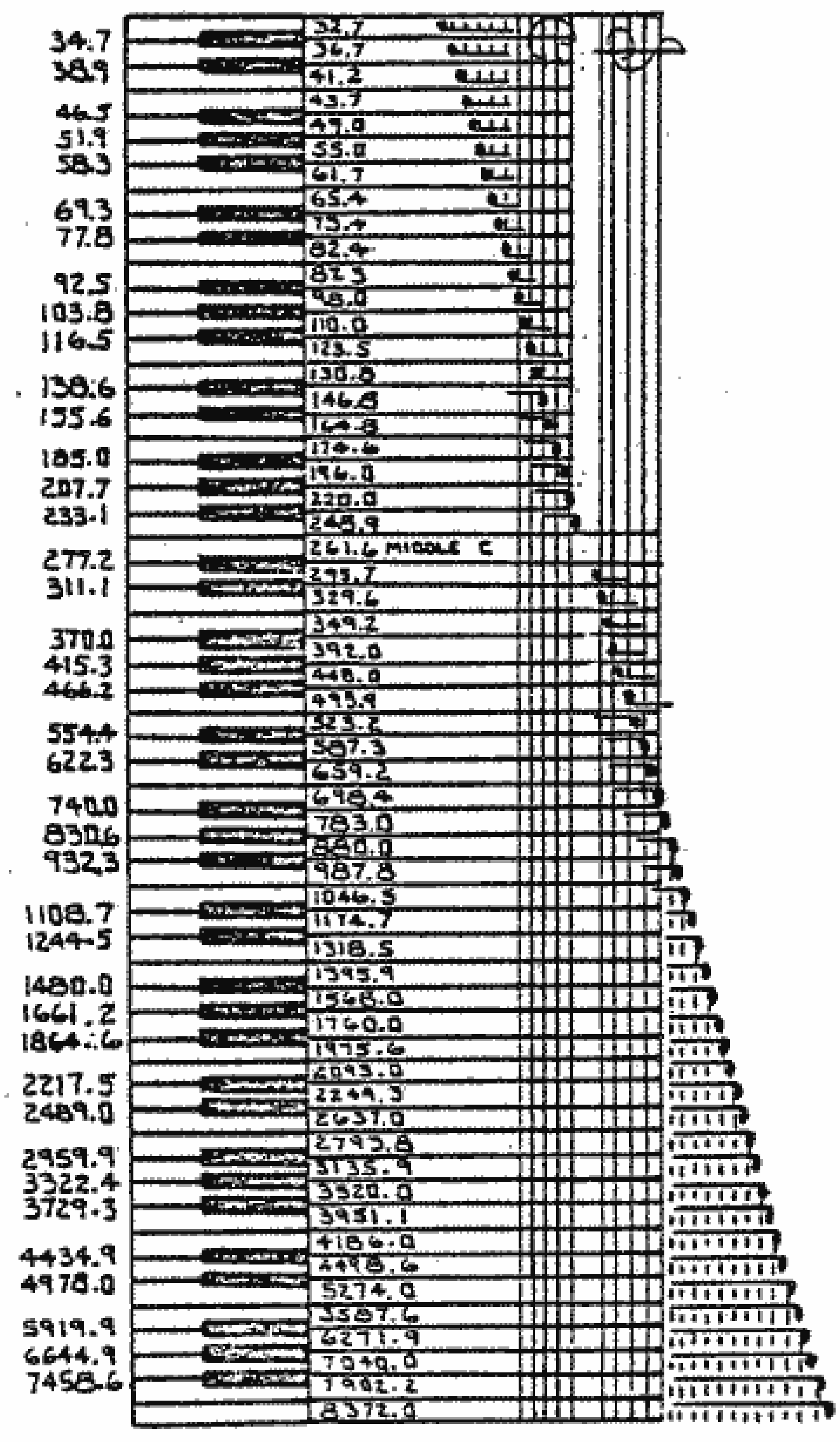
1. Star Trek
2. Cryptography
3. Black Jack
4. Hangman
5. 23 Matches

GD4

1. Frustration
2. Battleship
3. Tic-Tac-Toe
4. Civil War
5. Mastermind

Piano Keyboard

Frequency In Hertz



Votrax(R) Computer Voice Generation

To show the use of the voice generation feature, a demonstration disk is provided (Disk CA-14). To use the demonstration disk, the instructions included in the introduction to this manual will start the demonstration sequence, which will prompt you to select programs from a menu of choices.

<u>Creating Required Support Files</u>

After you have examined the demonstration programs, you will probably want to write your own user programs. Let's place the software we write on a disk we keep for software development. A copy of an Operating System Disk (OS-65D V3.1) will serve this purpose. Use the CREATE utility (discussed in "TO CREATE DISK FILES" in this manual) to CREATE a file on this disk, 4 tracks long, on which we can store the program we write. Let's name the file "TALK". Now, remove this personal software disk from the disk drive and store it away, temporarily.

Replace the demo disk (CA-14) in the disk drive. We wish, now, to load the program "TALKER" and its buffers. The program "TALKER" contains the computer's instructions to allow conversion from typed groups of characters to their sound equivalent (called phonemes). Boot up (restart) the system by pressing

<BREAK>

on your C4P keyboard. In response to

H/D/M?

type

<u>D</u>

The computer responds with a request to select a demonstration program from the demonstration disk, as

YOUR SELECTION?

We answer

PASS <RETURN>

Again, to the query

YOUR SELECTION?

only respond

<RETURN>

The computer responds

OK

We are now in the BASIC program and can load the desired files from the demonstration disk.

To load the program "TALKER" which provides the instructions to control the Votrax voice module, type

DISK! "LOAD TALKER" <RETURN>

We now have the Votrax controlling program (also called a device driver) in our program memory. Let's write a program to use the voice generation capability of our computer.

An Example Program

As a sample program, we wish to enter groups of characters which represent sounds (phonemes) from the keyboard. Then, we should like these sounds to be spoken by the Votrax voice generator. The phonemes which make up common English words are found in a dictionary in the Votrax Manual. For example, the word "hello" is found in the Votrax dictionary to be

PA1 PA1 H H H EH1 UH3 L UH3 O2 U1 U1 <RETURN>

Our program will accept character groups from the keyboard. We shall use spaces to separate these character groups. The end of a complete word or phrase will be designated by a <RETURN>. Once a complete character string is input from the keyboard, it will then be echoed by the Votrax module. The Votrax module is device #5.

The program to do these functions is

```
1Ø  PRINT"YOUR PHONEMES ARE" : INPUT A$ : ?#5,A$ : GOTO 1Ø
2Ø  END
RUN
```

The computer will prompt our response by typing

YOUR PHONEMES ARE?

Responding with the phonemes for the word "hello", we type

PA1 PA1 H H H EH1 UH3 L UH3 Ø2 U1 U1 <RETURN>

The word "hello" should be spoken by the Votrax module.

For a second word example, the Votrax dictionary lists "yes" with the possible phoneme representation

YES 1/PA1,2/Y,1/EH1,1EH3,1/S

To have the word "yes" spoken by the Votrax module, we would enter in response to

YOUR PHONEMES ARE?

the character string

PA1 Y Y EH1 EH3 S <RETURN>

The word "yes" should be spoken.

Storing the Sample Program

If you have finished with your sample program, we can store it in the previously CREATEd file, "TALK".

First, press

<RETURN>

to get back to BASIC. Then, your program can be placed on disk by typing

DISK!"PUT TALK" <RETURN>

We can assure ourselves that the program has been properly saved by restarting the system. When we get BASIC, type in response to

OK

the command

NEW <RETURN>

This will clear our BASIC work space. Now type

DISK!"LOAD TALK" <RETURN>

Then type

RUN

You should be back in your test program, which should function as before.

Appendix G

Disk Copy and Compare

Creating backup copies of your disks is a wise precaution. The backup copy provides protection against inadvertently destroying an important program, either by writing over the program or physically damaging the disk. Two utilities are provided for disk copying on your system disk.

Copying a disk requires two disk drives. (If you do not own a dual disk system, your OSI dealer can provide these services.) In a dual disk system, one drive will be labeled "A", the other drive will be labeled "B". Since we intend to overwrite material on one disk with material from another disk, extreme caution is urged in following the order of instructions. Otherwise, you can end up with two copies of the wrong disk!

First, select a disk on which to make a copy. This can be a new disk or a spare old disk. This disk should be initialized, a process of placing information on disk for timing purposes. Since this process will overwrite the entire disk, make sure this disk is truly available.

To initialize the disk, enter the operating system (From BASIC, we type EXIT). Place the disk ONTO WHICH you wish to make a copy in drive B. In response to the system prompt, type

A* SE B <RETURN>

Reply to the system response by

B* INIZ <RETURN>

The system will ask

ARE YOU SURE?

If you are sure, then type

YES <RETURN>

If any error message is reported, discard the disk as damaged or faulty. No errors will be reported for successful initialization. Now when the system prompt is shown, return to use A by replying

B* SE A

Before using your master disk, caution encourages us to cover the rectangular notch on the side of the disk with a piece of black electrical tape. This will "WRITE PROTECT" the disk against inadvertently overwriting data and programs we wish to keep. This tape may be removed later. Now we are ready to copy the master disk.

Place the master disk in drive A. The already initialized disk (ONTO which we copy) should be in drive B. CALL in the copy utility from disk by typing

A* CALL Ø2ØØ=13,1 <RETURN>

This will load the copy routine at location Ø2ØØ hex. To execute the copy routine, type

GO Ø2ØØ <RETURN>

You will be given the choice of

SELECT ONE:

1) COPIER

2) TRACK Ø READ/WRITE

Respond

?1 <RETURN>

to select the copier routine. (The TRACK Ø READ/WRITE is used to restore track Ø. This is typically needed if one powers down a disk drive with a disk in the drive.)

You will be asked

FROM DRIVE (A/B/C/D)?

Reply

A

The dialog continues:

TO DRIVE (A/B/C/D)?

Reply

B

Tracks are selected by replying

FROM TRACK?

by

Ø

TO TRACK (INCLUSIVE)?

39

Since we have proceeded carefully, the response to

ARE YOU SURE?

is

YES <RETURN>

Each track number, as it is copied, is displayed on the video screen.

Some OS-65D systems disks have a two sectored program called CONPAR on track 39.

A* CALL Ø2ØØ=39,1<RETURN>

A* CALL 2ØØØ=39,2<RETURN>

A* GO Ø2ØØ

The preceding sequence will call in the COMPAR routine as well as another version of the TRACKØ routine. The COMPAR utility is used in the same manner as the COPIER but tells you which bytes differ for the same location between drives A and B. This utility can be used to check copies for errors.

Appendix H

T E N A T I V E

Automatic Telephone Interface Application

The Automatic Telephone Interface provides a powerful feature for automatically dialing and answering the telephone. It accommodates both data and voice transmission, which permits your C4P to serve as a low cost data terminal or as a telephone answering service with equal ease. These individual functions would, in most systems, be served by separate modules at significantly higher cost.

Dialing is permitted in both the rotary dial (pulse) systems and the tone dialing systems. A simple modem would not provide this service normally.

By combining the voice, data, and dialing features, the C4P can place a telephone call automatically, request human aid, and proceed to transmit data to the devices selected by the person at the receiving end.

By having an automatic answering capability, the Automatic Telephone Interface can receive your call when you are away from home. By responding to dial tones sent from your location, tasks may be selected at home. These tasks may include playing back any recorded telephone messages, reporting on home security status, or starting your dinner and turning on the air conditioner. The combinations are almost limitless.

By incorporating direct telephone line connection, higher reliability and lower noise are obtained, while costs are held down on the interface by reducing complexity.

A program which utilizes many of these features follows. The program will initialize and set up the required dial codes.

By running this program and examining its listing, the combined power of the telephone interface and the home control features is evident. Automatic dialing and answering, combined with voice or data transmission and tone decoders, permits using your computer from a remote location. The use of dial tones as a key keeps your system secure from unauthorized use. The complete C4P's facilities can be used while retaining the privacy of the unit in your home.

```
1 REM TELCON TEST PROGRAM
10 DIMM(16),N$(16),D(16)
30 DATA 238,222,190,237,221,189,235,219,187,215
52 DATA 231,183,126,125,123,119
53 DATA 1,2,3,4,5,6,7,8,9,0,*,#,A,B,C,D
56 FORI=1TO16:READM(I):NEXTI
57 FORI=1TO16:READN$(I):NEXTI
100 A=62400:FORI=ATOA+6STEP2:POKEI,255:NEXTI
105 FORI=A+1TOA+7STEP2:POKEI,0:NEXTI
110 DATA 61,13,240,255
120 FOR I=ATOA+7STEP2:READB:POKE I,B:NEXTI
130 FORI=A+1TOA+7STEP2:POKEI,4:NEXTI:POKE62494,255
135 POKE 62496,3:POKE 62496,145:REM ACIA
140 PRINT"1 = ORIGINATE CALL PULSE DIALER
142 PRINT"2 = ORIGINATE CALL TONE DIALER
144 PRINT"3 = HANG UP"
145 PRINT"4 = DECODE TONE INPUTS"
146 PRINT"5 = END PROGRAM & HANG UP"
147 PRINT"6 = CALL NATIONAL WEATHER SERVICE"
148 PRINT"7 = CALL TIME SERVICE"
149 PRINT"8 = AUTO ANSWER MODE"
150 PRINT"9 = FORCE OFF HOOK AND YOUR FUNCTION"
200 INPUT"COMMAND";C
210 ON C GOTO 1000,1000,2000,9000,9999,3000,3050,4000,4030
220 GOTO 200
1000 REM ORIGINATE
1010 INPUT"NUMBER PLEASE";N$
1020 L=LEN(N$):E=1:POKE 62498,0
1030 FORJ=1TOL
1040 FORI=1TO16:IFMID$(N$,J,1)=N$(I)THEND(J)=M(I):E=0
1050 NEXTI:IF E=1 THEN GOTO 1010
1060 E=1:NEXTJ
1070 REM # IS NOW IN D(0) - D(15) - # OF DIGITS IN L
1080 POKE 62498,1:REM OFF HOOK
1090 IF C=1 THEN POKE 62400,33:REM PULSE DIAL
1100 IF C=2 THEN POKE 62400,29:REM TONE DIAL
1110 REM WAIT 1 SECOND
1115 D=60:GOSUB8000
1120 REM BEGIN DIAL
1130 FORI=1TOL:POKE62494,D(I):D=1:GOSUB8000:POKE62494,255:D=1
1160 GOSUB8000:NEXTI:GOTO140
2000 REM HANG UP PHONE
2010 GOSUB 9090:GOTO140
3000 A$="9311212":C=2:GOTO1020
3050 A$="4711212":C=2:GOTO1020
4000 REM ANSWER MODE
4010 PRINT"3 = TAPE REC."
4015 PRINT"4 = MODEM RCV."
4020 PRINT"5 = TONE DECODER"
4030 INPUT"YOUR INPUT FUNCTION";R
4060 PRINT"0 = NO OUTPUT"
4065 PRINT"1 = VOTRAX OUTPUT"
4070 PRINT"2 = AUX OUTPUT"
4075 PRINT"3 = TAPE PLAYER OUTPUT"
4080 PRINT"4 = MODEM OUTPUT"
4090 PRINT"5 = TONE GEN. OUTPUT"
4094 INPUT"OUTPUT FUNCTION";X
4100 POKE 62402,((R*8)+X)
4105 IF C<>9 THEN GOTO 4120
4110 Z=PEEK(62403):Z=ZAND128:IF Z=0 THEN 4110
4120 GOSUB 9080
4130 IF R=3 GOTO 140
4132 IF R=4 THEN PRINT"INSERT YOUR MODEM PROGRAM":GOTO140
4134 IF R=5 GOTO 9001
4140 GOTO 4010
8000 FORT=1TOD:WAIT 56832,128,128:WAIT 56832,128:NEXTT:RETURN
9000 GOSUB 9080:POKE 62402,45
9001 PRINT:PRINT:PRINT"PRESS (8) TO EXIT THIS MODE & HANG UP"
9005 Z=PEEK(62493):Z=ZAND128:IFZ=0 THEN 9020
9010 GOTO9005
9020 F=PEEK(62492) AND 15:REM GET DATA AND RESET FLAG
9030 PRINT"YOU PRESSED ";N$(F):IFN$(F)="8"THENGOSUB9090:GOTO140
9040 GOTO9005
9080 POKE 62498,1:RETURN:REM LIFT HOOK
9090 POKE 62498,0:RETURN
9999 POKE 62498,0:END
```

T E N A T I V E

Automatic Telephone Interface, Hardware

Detailed use of this very powerful feature will require more detailed information on addresses dedicated to this unit.

This information is incorporated in the BASIC program in the previous section. Extracting the part of the program you find useful is the easiest way to generate a program. However, the details are given for reference.

The standard Touch-Tone(C) keypad

Row \ Column	1	2	3	4
1	1	2	3	A
2	4	5	6	B
3	7	8	9	C
4	*	0	#	D

Note that column 4 is only available on special telephones.

is shown above, divided into rows and columns.

To generate the tones corresponding to these keys, using the tone generator (or activate the pulse dialer on rotary dial systems), the PIA data port at F8Ø6 must be given the bit pattern below:

	Column and Row									
	C1	C3	C2	C1	R4	R3	R2	R1	PIA Data Value (Hex)	PIA Data Value (Dec.)	Key
PIA Bit:	1PB7	1PB6	1PB5	1PB4	1PB3	1PB2	1PB1	1PBØ			
	1	1	1	Ø	1	1	1	Ø	EE	238	1
	1	1	1	Ø	1	1	Ø	1	ED	237	4
	1	1	1	Ø	1	Ø	1	1	EB	235	7
	1	1	1	Ø	Ø	1	1	1	E7	231	*(1)
	1	1	Ø	1	1	1	1	Ø	DE	222	2
	1	1	Ø	1	1	1	Ø	1	DD	221	5
	1	1	Ø	1	1	Ø	1	1	DB	219	8
	1	1	Ø	1	Ø	1	1	1	D7	215	Ø
	1	Ø	1	1	1	1	1	Ø	BE	19Ø	3
	1	Ø	1	1	1	1	Ø	1	BD	189	6
	1	Ø	1	1	1	Ø	1	1	BB	187	9
	1	Ø	1	1	Ø	1	1	1	B7	183	#(1)
	Ø	1	1	1	1	1	1	Ø	7E	126	A(1)
	Ø	1	1	1	1	1	Ø	1	7D	125	B(1)
	Ø	1	1	1	1	Ø	1	1	7B	123	C(1)
	Ø	1	1	1	Ø	1	1	1	77	119	D(1)
	1	1	1	1	1	1	1	1	FF	255	"OFF"(2)

(1) These keys exist on Touch-Tone(C) sets only.

(2) "OFF" is used for space between tones. It does not have a corresponding key. A space of $\geq$ 35 ms must exist between tones. Tone duration must be $\geq$ 33 ms. *

*Note: These timing values are subject to modification.

We use the PIA practice of locating the control register one location higher than the port it controls (PIA tone generator port = F8Ø6, tone generator control port = F8Ø7).

Each of the other PIA ports serves multiple functions, with each bit serving to choose or exclude a particular function. In order of locations these PIA's and their functions are:

Address

Decimal	Hex	Bit PA2	PA1	PAØ	Value of PA2 PA1 PAØ	Function, which line is selected. All lines are output (from CPU)
63488	F8ØØ					
		Ø	Ø	Ø	Ø	Spare line out to phone
		Ø	Ø	1	1	Votrax Module
		Ø	1	Ø	2	Auxiliary Device (Digital to Analog Converter, DAC)
		Ø	1	1	3	Tape Recorder Play
		1	Ø	Ø	4	Modem
		1	Ø	1	5	Tone Dialer Generator
		1	1	Ø	6	Pulse Dialer
		1	Ø	1	7	Spare Line
						Note: Select #7 when you are not outputting to the phone

Address

Decimal	Hex	Bit PA5	PA4	PA3	Value of PA5 PA4 PA3	Description of input (to CPU) selected
63488	F8ØØ					
		Ø	Ø	Ø	Ø	Spare line from phone
		Ø	Ø	1	1	Line decode
		Ø	1	Ø	2	Auxiliary Device (Analog to Digital Converter, ADC)
		Ø	1	1	3	Tape Recorder Record
		1	Ø	Ø	4	Modem
		1	Ø	1	5	Tone Decoder
		1	1	Ø	6	Spare
		1	1	1	7	Spare

PA6 = Spare

PA7 = Ring sense (1 = Ring, Ø = No Ring)
(input to CPU)

63489 F8Ø1 Control register for 63488 decimal.

Address

Decimal	Hex	Bit	Use
6349Ø	F8Ø2		
		PBØ	- Connect control (1 = true) (output from CPU)
		PB1	- Connect control (Ø = true) (output from CPU)
		PB2	- Tape input control (output from CPU)
		PB3	- Tape output control (output from CPU)
		PB4	- Modem Mode Sense (1 = originate, Ø = answer) (input to CPU)
		PB5	
		PB6	- Unused
		PB7	

63491 F8Ø3 Control register for 6349Ø

Address Decimal 63492	Hex F8Ø4	Bits (See Note 1) 1PA3	1PA2	1PA1	1PAØ	Hex Value 1PA3 1PA2 1PA1 1PAØ	Tone Input From Phone (input to CPU)
		Ø	Ø	Ø	Ø	Ø	D
		Ø	Ø	Ø	1	1	1
		Ø	Ø	1	Ø	2	2
		Ø	Ø	1	1	3	3
		Ø	1	Ø	Ø	4	4
		Ø	1	Ø	1	5	5
		Ø	1	1	Ø	6	6
		Ø	1	1	1	7	7
		1	Ø	Ø	Ø	8	8
		1	Ø	Ø	1	9	9
		1	Ø	1	Ø	A	Ø
		1	Ø	1	1	B	*
		1	1	Ø	Ø	C	#
		1	1	Ø	1	D	A
		1	1	1	Ø	E	B
		1	1	1	1	F	C

CA1 - 1 = valid tone decode

Ø = no tone

Note:

1) PAØ - PA3 must be read within 25 ms of CA1 going high (i.e., IRQA1=1)

2) CA1 must be continuously read at least every 33 ms (IRQA1) when listening for tones.

3) Software should program control for low to high transitions of CA1.

Address			
Decimal	Hex	Bits	Function
63492	F8Ø4	1PA4 -	Modem self-test (CPU output)
		1PA5 -	Modem squelch (CPU output)
		1PA6 -	Modem originate mode ($\overline{SH}$) (CPU output)
		1PA7 -	Modem answer mode ($\overline{RI}$) (CPU output)
		1CA1 -	DTMF decodes strobe (Input to CPU) This line is polled to determine whether data on 1PAØ to 1PA3 is valid.
63493	F8Ø5	Control register for 63492	
63494	F8Ø6	1PBØ - 1PB7	Dialer data (See page 195)
		1CB1, 1CB2	Unused
63495	F8Ø7	Control register for 63494	
63496	F8Ø8	ACIA	
63497	F8Ø9	Control register for 63496	

A more detailed connection discussion with schematic diagrams accompanies the Automatic Telephone Interface. Having the signal data presented here, the versatility and power of the interface can be fully utilized and the programming needs anticipated. Reference to popular books on telephone company notation, such as J. Sunier's <u>The Handbook of Telephones and Accessories</u> (Tab Books, Blue Ridge Summit, PA 17214) are recommended reading to support detailed communications plans.

Appendix I

Hex to Decimal Tutor

Within computers, calculations are made in zeros and ones, a binary system. This representation of numbers is more convenient than on traditional base 1Ø (decimal) system. For compact notation, the binary representation is often written by grouping multiples of 2, specifically powers of 2*2*2*2=16. This notation, base 16 is called a hexadecimal number system.

We can use the manual's illustrations of the ASC and CHR$ commands to write a program to convert decimal numbers (counting in base 1Ø) to hexadecimal numbers (counting in base 16).

To count in our base 1Ø numbering system, we use the symbols Ø, 1, 2, 3, 4, 5, 6, 7, 8, and 9. We need 1Ø symbols in all. We use a place holder notation to represent a number, so that

123 = 1*1Ø^2 + 2*1Ø^1 + 3*1Ø^Ø = 1ØØ + 2Ø +3

= 1*1ØØ + 2*1Ø + 3*1

= 1ØØ + 2Ø + 3

(where ^ indicates "to the power).

As our other case, base 16 (hexadecimal) counting will require 16 symbols. By common agreement the symbols are Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Here A hexadecimal corresponds to 1Ø decimal, B hexadecimal corresponds to 11 decimal, etc. Therefore, the number

123 hexadecimal = 1*16^2 + 2*16^1 + 3*16^Ø decimal

= 1*256 + 2*16 + 3*1 decimal

= 256 + 32 + 3 decimal = 291 decimal

Similarly, the number 3A hexadecimal is

3A hexadecimal = 3*16^1 + 1Ø*16^Ø decimal

= 3*16 + 1Ø*1 decimal

= 48 + 1Ø decimal

= 58 decimal

This much calculation is a sure candidate for a computer program. Also, in some of the advanced programming techniques, we shall want to be able to convert from one system to another. This problem of number system conversion gives us a chance to use the ASCII conversion commands in our programming. Moreover, this program is readily modified to permit data entry into programs in either hexadecimal or decimal. For occasional conversions, we also provide a decimal to hexadecimal conversion table elsewhere in the appendix. Let's look at the ASCII code table on page

Symbols Ø through 9 have ASCII codes of 48 to 57 decimal. By subtracting 48 from this ASCII decimal code, we can get the numbers value in the range Ø to 9. For example, the ASCII code for 3 is given as:

ASCII Code for symbol "3" = 51

If we subtract 48 from 51 (the ASCII code value of the number 3), we get the numeric value, 3.

ASCII code for symbol "3" = 48 = 51-48 = 3

This observation permits us to change the code representation of numbers Ø to 9 into the numbers, themselves.

Similarly, we see the symbols A to F are represented by ASCII codes of 65 to 7Ø decimal. By subtracting 55 from this code, we can get the decimal value which the hexadecimal notation implies.

That is, the observations

1) the ASCII code for the symbol "A" = 65

2) the number A hexadecimal = 1Ø decimal

3) thus the ASCII code for "A" - 55 = 65-55 = 1Ø

permit us to convert values for the ASCII symbols A to F. We can use this conversion to complete our algorithm for conversions from hexadecimal to decimal.

To go from decimal to hexadecimal (the reverse direction), we note how remainders from division yield the separate digit's representation. For example, in base 1Ø, for the number 123, try successive divisions, and observe the remainder

```
1Ø / 123
1Ø /  12 + remainder 3
1Ø /   1 + remainder 2
       Ø + remainder 1
```

which yields the base 1Ø representation when read in the direction of the arrow. Trying this in base 16 to find the hexadecimal value of 2Ø decimal

```
16 / 2Ø
16 /  1 + remainder 4
      Ø + remainder 1
```

gives the hexadecimal value of 14 when read in the direction of the arrow. This checks since

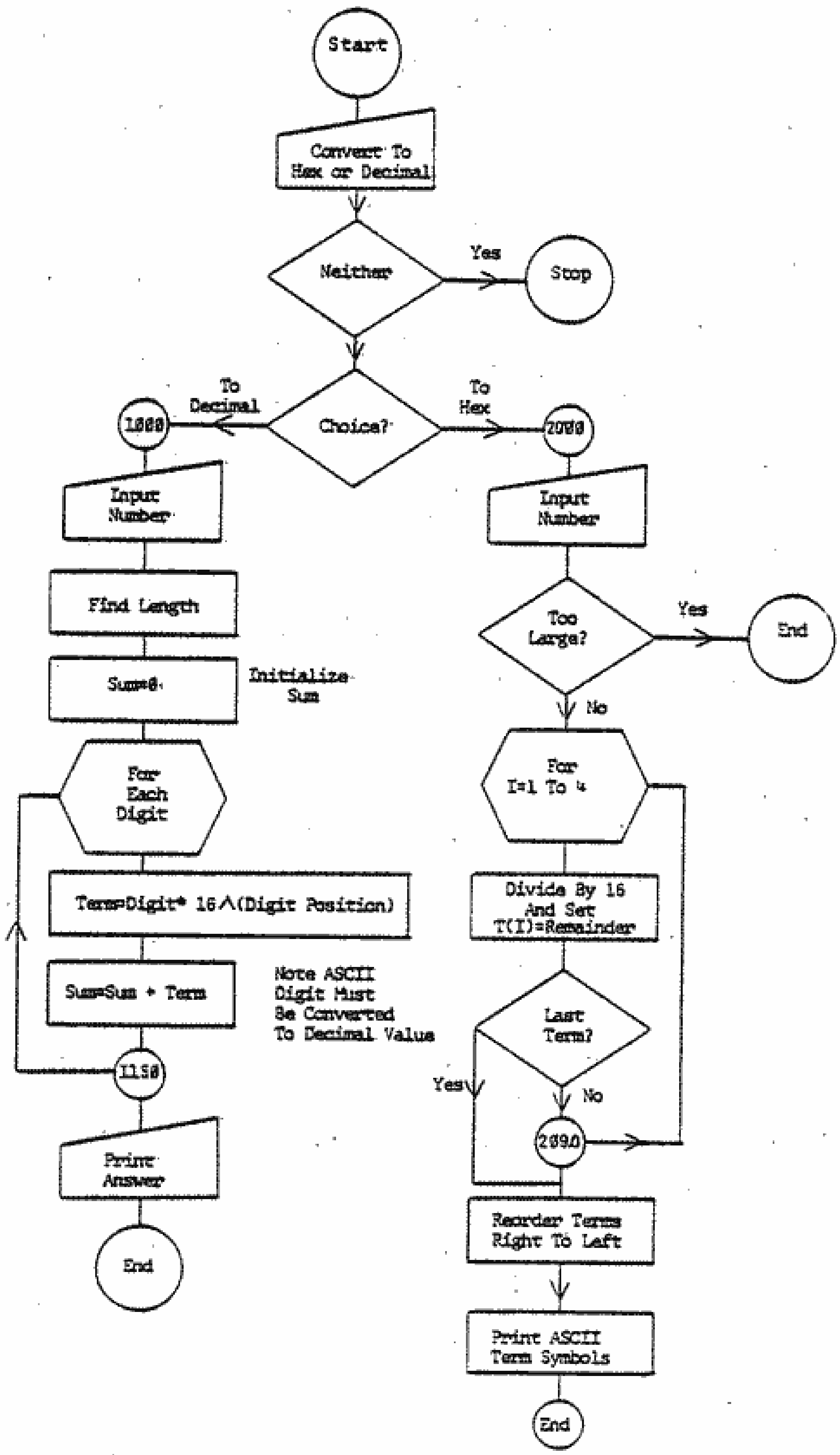
```
1*16^1 + 4*16^Ø = 2Ø
```

Slightly harder is converting 28 decimal

```
16 / 28
16 /+ 1 + remainder 12 = B hexadecimal
      Ø + remainder 1 - 1 hexadecimal
```

giving the hexadecimal value of 1B.

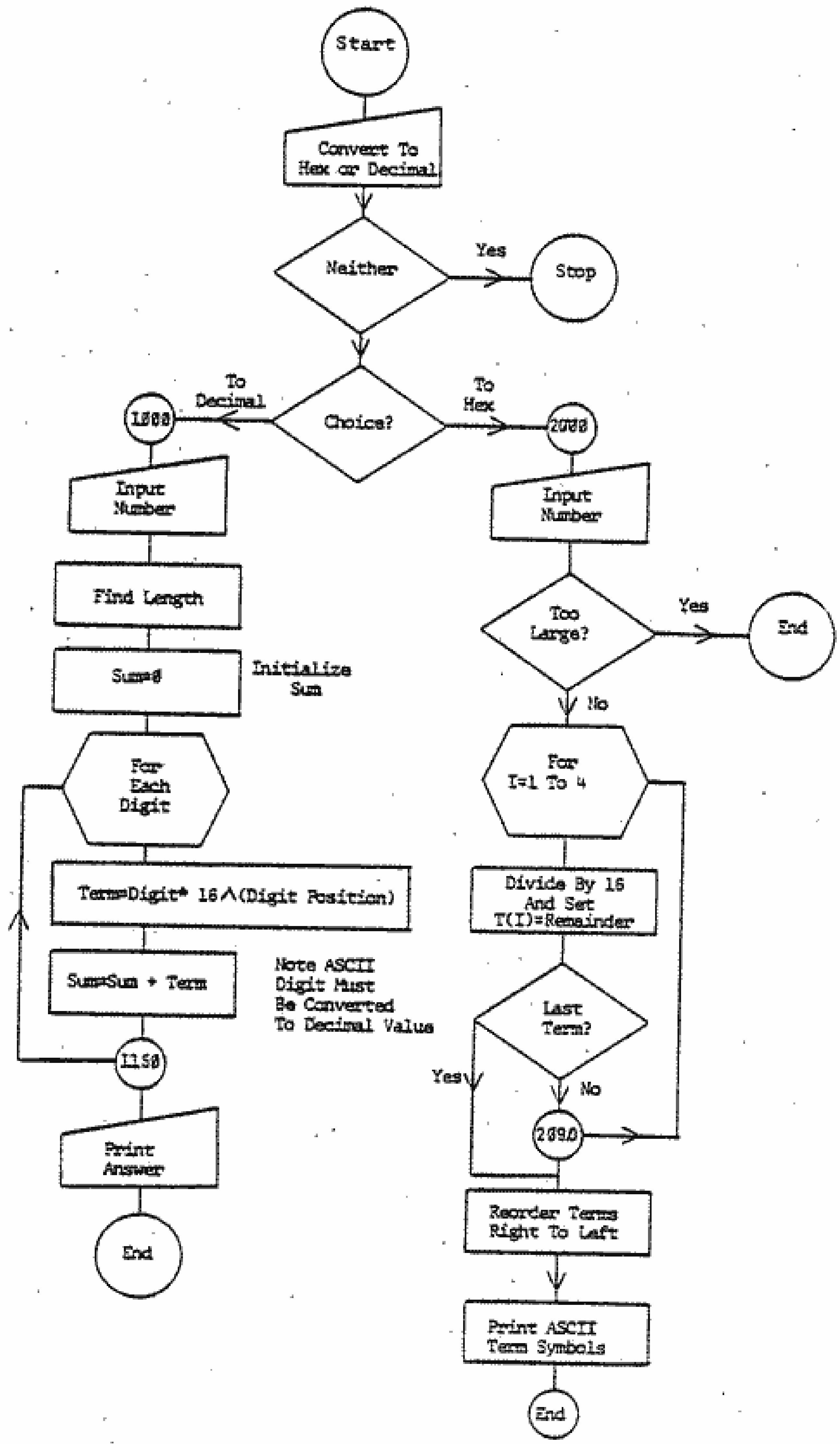
Let's put these two conversion algorithms together in a flow chart, shown here in overall form.



```
10 REM HEX AN OSI PROGRAM TO CONVERT
20 REM     1)  HEXADECIMAL (BASE 16) TO DECIMAL OR
30 REM     2)  DECIMAL TO HEXADECIMAL: L  ROEMER 28 MAY 1979
35 PRINT" TYPE ":PRINT"    1 FOR HEX TO DECIMAL
36 PRINT"    2 FOR DECIMAL TO HEX"
40 INPUT "YOUR CHOICE IS ";CHOICE
50 IF CHOICE=1 THEN GOSUB 1000: REM HEX TO DECIMAL
60 IF CHOICE=2 THEN GOSUB 2000: REM DECIMAL TO HEX
70 IF CHOICE<>1 AND CHOICE <>2 THEN GOSUB 3000
80 END
100 REM CONVERT EACH CHARACTER TO ASCII CODE
1000 REM   HEX INPUT TO DECIMAL OUTPUT
1010 INPUT "YOUR HEX NUMBER IS ";A$
1020 L=LEN(A$)
1030 SUM=0
1040 REM     WHEN EXAMINE CHARACTERS, LOW POSITION
1050 REM     IS AT RIGHT HAND
1060 FOR  K=1 TO L
1070 M=L+1-K
1080 T2=ASC(MID$(A$,M,1))
1100 S1=SUM+16^(K-1)*(T2-55)
1110 S2=SUM+16^(K-1)*(T2-48)
1130 IF T2>64 THEN SUM=S1:REM CHECK IF HEX CHAR>9
1140 IF T2<64 THEN SUM=S2:REM           OR <9
1150 NEXT K
1160 PRINT "DECIMAL VALUE IS ";SUM
1170 RETURN
1180 END
2000 REM    DECIMAL INPUT WITH HEX OUTPUT
2010 INPUT "YOUR DECIMAL IS  ";D
2020 IF D>65535 THEN GOTO 2600
2030 T(0)=D
2040 FOR I=1 TO 4
2050 T(I)=INT(T(I-1)/16)
2060 CI(I)=T(I-1)-T(I)*16
2070 K=I
2080 IF INT(T(I))=0 THEN GOTO 2200
2090 NEXT I
2200 FOR I=1 TO K
2210 REM:     REVERSE ORDER OF DIGITS FOR PRINTING
2220 CH$(K+1-I)=CHR$(48+CI(I))
2230 IF CI(I)>9 THEN CH$(K+1-I)=CHR$(55+CI(I))
2240 NEXT I
2250 ZIPS$=" "
2260 FOR I=1 TO K
2270 ZIP$=ZIP$+CH$(I)
2280 NEXT I
2290 PRINT "HEX   ";ZIP$
2300 RETURN
2310 END
2600 PRINT "TOO LARGE A VALUE"
2610 END
3000 PRINT "YOUR CHOICE  SHOULD BE 1 OR 2"
3010 PRINT "RUN AGAIN IF YOU WISH CHOICE"
3020 RETURN
3030 END
```

giving the hexadecimal value of 1B.

Let's put these two conversion algorithms together in a flow chart, shown here in overall form.



HEXADECIMAL—DECIMAL CONVERSION

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
010	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
020	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
030	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
040	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
050	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
060	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
070	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
080	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
090	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
0A0	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
0B0	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
0C0	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
0D0	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
0E0	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
0F0	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
110	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287
120	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303
130	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319
140	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335
150	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351
160	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367
170	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383
180	3E4	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399
190	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415
1A0	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431
1B0	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447
1C0	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463
1D0	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479
1E0	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495
1F0	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511
220	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559
230	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575
240	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591
250	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607
260	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623
270	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639
280	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655
290	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671
2A0	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687
2B0	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703
2C0	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719
2D0	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735
2E0	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751
2F0	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767

HEXADECIMAL-DECIMAL CONVERSION

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
300	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783
310	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799
320	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815
330	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831
340	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847
350	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863
360	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879
370	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895
380	896	897	858	899	900	901	902	903	904	905	906	907	908	909	910	911
390	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927
3A0	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943
3B0	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959
3C0	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975
3D0	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991
3E0	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1516	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

HEXADECIMAL-DECIMAL CONVERSION

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

HEXADECIMAL—DECIMAL CONVERSION

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

HEXADECIMAL–DECIMAL CONVERSION

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839

13	TrackØ/Copier utility (loads to $Ø2ØØ for 5 pages).
14 - 38	User programs and OS-65D utility BASIC programs.
39	Compare routine, on some disks only.

I/O Flag Bit Settings

INPUT:

BIT Ø - ACIA on CPU board (terminal).
BIT 1 - Keyboard on 54Ø board.
BIT 2 - UART on 43Ø/55Ø board.
BIT 3 - NULL.
BIT 4 - Memory input (auto incrementing).
BIT 5 - Memory buffered disk input.
BIT 6 - Memory buffered disk input.
BIT 7 - 55Ø board ACIA input. As selected by index at location $2323 (8995 decimal).

OUTPUT:

Bit Ø - ACIA on CPU board (terminal).
Bit 1 - Video output on 54Ø board.
Bit 2 - UART on 43Ø/55Ø board.
Bit 3 - Line printer interface.
Bit 4 - Memory output (auto incrementing).
Bit 5 - Memory buffered disk output.
Bit 6 - Memory buffered disk output.
Bit 7 - 55Ø board ACIA output. As selected by index.

9 Digit BASIC Extensions

INPUT PNDSGN(DEVICE NUMBER)	(input is set to new device, output is set to null device. If device number > 3, and null inputs are ignored if device number > 3.)
INPUT "TEXT";PNDSGN(DEVICE NUMBER)	(print "TEXT" at current output device, then function as above).
PRINT PNDSGN(DEVICE NUMBER):	(print output for this command at new device).
LIST PNDSGN(DEVICE NUMBER)	(list program or segments of program to new device).

WHERE (DEVICE NUMBER) FOR OUTPUT IS:

1 - ACIA terminal
2 - 54Ø video terminal
3 - 43Ø/550 ACIA UART port
4 - Line printer
5 - Memory output
6 - Memory buffered disk output (bit 5).
7 - Memory buffered disk output (bit 6).
8 - 55Ø ACIA output
9 - Null output

(DEVICE NUMBER) FOR INPUT IS:

1 - ACIA terminal
2 - 54Ø keyboard
3 - 43Ø/55Ø ACIA UART port
4 - Null device
5 - Memory input
6 - Memory buffered disk input (bit 5).
7 - Memory buffered disk input (bit 6).
8 - 55Ø ACIA input
9 - Null Input

AND WHERE PNDSGN IS A POUND SIGN.

EXIT	Exit to OS-65D V3.Ø
RUN (STRING)	Load and run file with name in (STRING).
DISK ! (STRING)	Send (STRING) to OS-65D V3.Ø as a command line.
DISK OPEN,(DEVICE),(STRING)	Open sequential access disk file with file name, (STRING) using memory buffered disk I/O distributor device number 6 or 7. Reads first track of file to memory and sets up the memory pointers to start of buffer.
DISK CLOSE,(DEVICE)	Forces a disk write of the current buffer contents to current track.
DISK GET,(RECORD NUMBER)	Using last file opened on the LUN 6 device, a calculated track is read into memory. Where that track is: INT(REC. NUM)/24+base track given in last open command.
DISK PUT	It also sets both memory pointers to: 128*(REC. NUM.)-INT(REC. NUM.)/24))+ base buffer address for LUN 6 device. Write device 6 buffer out to disk. The effect is the same as a "DISK CLOSE,6".

Extensions to Assembler

E	Exit to OS-65D V3.
H(HEX NUM)	Set high memory limit to (HEX NUM).
M(HEX NUM)	Set memory offset for A3 assembly to (HEX NUM).
!(CMD LINE)	Send (CMD LINE) to OS-65D V3 as a command to be executed and then return to assembler.
CONTROL-I	Tab 8 spaces. Also: CONTROL-U 7 spaces. CONTROL-Y 6 spaces. CONTROL-T 5 spaces. CONTROL-R 4 spaces. CONTROL-E 3 spaces.
CONTROL-C	Abort current operation.

Extended Monitor

!TEXT	Sent "TEXT" to OS-65D V3 as a command.
@NNNN	Open memory location "NNNN" for examination. Subcommands: LF - Open next location. CR - Close location. DD - Place "DD" into location. " - Print ASCII value of location. / - Reopen location. Uparrow - Open previous location.
A	Print AC from breakpoint.
BN,LLLL	Place breakpoint "N" (1-8) at location, "LLLL".
C	Continue from last breakpoint.
DNNNN,MMMM	Dump memory from "NNNN" to "MMMM".
EN	Eliminate breakpoint "N".
EXIT	Exit to OS-65D V3.Ø.
FNNNN,MMMM=DD	Fill memory from "NNNN" to "MMMM"-1 with "DD".
GNNNN	Transfer control to location "NNNN".
HNNNN,MMMM(OP)	Hexadecimal calculator prints result of "NNNN"(OP)"MMMM" where (OP) is + - * /.

I	Print break information for last breakpoint.
K	Print stack pointer from breakpoint.
L	Load memory from cassette.
MNNNN=MMMM,LLLL	Move memory block "MMMM" to "LLLL"-1 to location "NNNN" and up in memory.
NHEX)NNNN,MMMM	Search for string of bytes "HEX" (1-4) between memory location "NNNN" and "MMMM"-1.
O	Print overflow/remainder from hex calculator.
P	Print processor status word from breakpoint.
QNNNN	Disassemble 23 lines from location "NNNN". A linefeed continues disassembly for 23 more.
RMMMM=NNNN,LLLL	Relocate "NNNN" to "LLLL"-1 to location "MMMM"
SMMMM,NNNN	Save memory block, "MMMM" to "NNNN"-1 on cassette.
T	Print breakpoint table.
V	View contents of cassette.
WTEXT)MMMM,NNNN	Search for ASCII string "TEXT" between "MMMM" and "NNNN"-1
X	Print X index register from last break.
Y	Print Y index register from last break.

NOTE: All commands are line buffered by OS-65D. Thus only 18 characters per line are allowed and CONTROL-U and BACKARROW apply.

Source File Format

Relative Disk Address	Memory Address	Usage
Ø	$3279	Source start (low)
1	$327A	Source start (high)
2	$327B	Source end (low)
3	$327C	Source end (high)
4	$327D	Number of tracks req.
5 and on ...	$327E and on..	Source text

Directory Format

Two sectors (1 and 2) on track 12 hold the directory information. Each entry requires 8 bytes. Thus there are a total of 64 entries between the two sectors. The entries are formatted as follows:

Ø - 5	ASCII 6 character name of file
6	BCD first track of file
7	BCD last track of file (included in file).

Track Formatting

The remaining tracks are formatted as follows:

- 1Ø millisecond delay after the index hole
- a 2 byte track start code, $43 $57
- BCD track number
- a track type code, always a $58

There can be any mixture of various length sectors hereafter. The total page count cannot exceed 8 pages if more than one sector is on any given track.

Each sector is written in the following format:

- previous sector length (4 if none before) times 8ØØ microseconds of delay
- sector start code, $76
- sector number in binary
- sector length in binary
- sector data

Diskette Copier

The diskette copy utility is found on track 1 sector 2. It should be loaded into location 2ØØ with a "CA Ø2ØØ=13,1. To start it type, "GØ 02ØØ". To select the copier type a "1". The copier automatically formats the destination diskette before writing on it.

Track Ø Read/Write Utility

This utility permits the reading of data on track Ø anywhere into memory. Also the capability is available to write any block of memory to track Ø specifying a load address and page count. The track zero format is as follows:

- 10 millisecond delay after the index hole
- the load address of the track in high-low form
- the page count of how much data is on track zero

End User POKEs to BASIC

Location	Old	New	Function
2972	58	13	Disable , and : terminators on string input
2976	44	13	
2Ø73	173	96	Ignore CONTROL-C
2893	55	28	Disable break on null input.
2894	Ø8	11	"REDO FROM START"
741	76	1Ø	Remove keywords, "NEW" and "LIST"
75Ø	78	1Ø	

Other POKEs to BASIC

Location	Function
23	Terminal width
2888,8722	If both are Ø a null input to a "INPUT" statement yields an empty string or a Ø. If both are 27 then the input statement functions are normal.
8917	USR(X) Disk Operation Code: Ø - write to Drive A 3 - read from Drive A 6 - write to Drive B 9 - read from Drive B

9826	Track number for USR(X) Disk Operation
9822	Sector number for USR(X) Disk Operation
9823	Page count for USR(X) disk write, or number of pages read in by disk read
9824	Low byte of address of memory block for USR(X) Disk Operation
9825	High byte of address of memory block for USR(X) Disk Operation
8954	Location of JSR to a USR function. Preset to JSR $22D4, i.e., set up for USR(X) Disk Operation
8993	I/O Distributor INPUT flag
8994	I/O Distributor OUTPUT flag
8995	Index to current ACIA on 55Ø board. If numbered from Ø to 15 the value POKEd here is 2 times the ACIA number.
8996	Location of a random number seed. This location is constantly incremented during keyboard polling.
896Ø	Has page number of highest RAM location found on OS-65D's cold start boot in. This is the default high memory address for the assembler and BASIC
9Ø98	Low byte address for memory input
9Ø99	High byte address for memory input
91Ø5	Low byte address for memory output
91Ø6	High byte address for memory output
9132	Low byte address for memory buffered disk input
9133	High byte address for memory buffered disk input Bit 5 device. Defaults to $327E
9155	Low byte address for memory buffered disk output
9156	High byte address for memory buffered disk output Bit 5 device. Defaults to $327E.
9213	Low byte address for memory buffered disk input
9214	High byte address for memory buffered disk input Bit 6 device. Defaults to $3A7E.
9238	Low byte address for memory buffered disk output

9239	High byte address for memory buffered disk output. Bit 6 Device. Defaults to $3A7E.
8998	Memory buffered disk I/O Bit 5 Device Parameters: 8998-8999 - Buffer start address ($327E) 9ØØØ-9ØØ1 - Buffer end address ($3A7E) 9ØØ2 - First track of file 9ØØ3 - Last track of file 9ØØ4 - Current track in buffer 9ØØ5 - Dirty buffer flag (Ø=clean)
9ØØ6	Memory buffered disk I/O bit 6 Device Parameters: 9ØØ6-9ØØ7 - Buffer start address ($3A7E) 9ØØ8-9ØØ9 - Buffer end address ($427E) 9Ø1Ø - First track of file 9Ø11 - Last track of file 9Ø12 - Current track in buffer 9Ø13 - Dirty buffer flag (Ø=clean)
12Ø42	Location of the 24 used by the random access file calculation routines. This location should only be altered after the open has occurred for the random access file because the PUT GET code is loaded into the directory buffer. This is where this 24 resides. Making it a 48 gives one 64 byte records.
9368	High byte address for indirect file input (low=ØØ)
9554	High byte address for indirect file output (low=ØØ)